AFIT/DS/ENG/93-02

A DECISION CRITERIA TO SELECT AN ASSOCIATIVE-MEMORY ORGANIZATION THAT MINIMIZES THE EXECUTION TIME OF A MIX OF ASSOCIATIVE-SEARCH OPERATIONS

DISSERTATION

David W. Banton, Captain, USAF

AFIT/DS/ENG/93-02

AFIT/DS/ENG/93-02

A DECISION CRITERIA TO SELECT AN ASSOCIATIVE-MEMORY ORGANIZATION THAT MINIMIZES THE EXECUTION TIME OF A MIX OF ASSOCIATIVE-SEARCH OPERATIONS

DISSERTATION

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

David W. Banton,

Captain, USAF

June 1993

AFIT/DS/ENG/93-02

A DECISION CRITERIA TO SELECT AN ASSOCIATIVE-MEMORY ORGANIZATION THAT MINIMIZES THE EXECUTION TIME OF A MIX OF ASSOCIATIVE-SEARCH OPERATIONS

David W. Banton,

Captain, USAF

Approved:

	1 June 93
	1 June 93
	1 June 93
	1 June 93

Accepted:

Dean, School of Engineering

Preface

This dissertation would not have been possible without the help of several individuals. Major Mehalic's guidance and advice were invaluable throughout the research. Also, his technical expertise on VLSI design issues greatly improved the quality of the simulation models used in this research. Lt Col Hobart's technical expertise in the field of computer architectures and meticulous evaluation of the dissertation greatly improved this endeavor. Dr Oxley's help with the mathematical derivations and equations improved the quality of the dissertation.

Of all the people involved in this research, my family made the greatest sacrifices and the most important contributions. My wife, Carol, kept the household running, so I could have the time to complete my research. My children, Sarah, John, and Daniel accepted my long work hours with little complaining. Though the degree is awarded to a single individual, it as much theirs as mine.

Table of Contents

Page

List of Figure x

List of Tables xiii

Abstract xvii

I. Introduction 1

1.1 Introduction 1

1.2 Background 1

1.3 Purpose 4

1.4 Approach 5

1.5 Overview 7

II. Theoretical Foundation Development 9

2.1 Introduction 9

2.2 Associative-Search Classification 9

2.2.1 The Boolean Algebra 10

2.2.2 Feng's Associative-Search Operations 13

2.3 Reclassification of Feng's Associative-Search Operations 16

2.4 A Relationship Between the Hardware-Influenced Associative-Search Operation Categories and the Associative-Memory Organizations 17

2.4.1 CAM Description 18

2.4.2 BSWPAM Organization Description 20

2.4.3 ESAM Organization Description 23

2.4.4 Associative-Memory Execution of a Mix of Associative-Search Instructions 24

2.5 Summary 27

III. Static Random-Access Memory 28

3.1 Introduction 28

3.2 SRAM-Organization Description 28

3.3 SRAM-Architecture Description 29

3.3.1 Data-Write Circuit Description 31

3.3.2 Address-Driver Description 33

3.3.3 Address-Decode Circuit Description 36

3.3.4 SRAM-Array Description 40

3.3.5 Data-Read Circuit Description 42

3.4 Read and Write-Instruction Access-Time Characteristic Equations ... 46

3.5 Summary 49

IV. Word-Organized Fully-Parallel Associative Memory 50

4.1 Introduction 50

4.2 CAM Organization and General Operation Description 50

4.3 CAM-Architecture Description 52

4.3.1 Data-Write Circuit Description 53

4.3.2 CAM-Array Description 55

4.3.3 Data-Read Circuit Description 60

4.3.4 Word-Select Circuit Description 62

4.3.4.1 Word Circuit and Word\Search Port Drivers Description 66

4.3.4.2 Match-Circuit Description 69

4.3.4.3 MMR-Circuit Description 71

4.4 CAM-Operation Description 71

4.4.1 Word Operation Mode 71

4.4.1.1 Write Instructions 72
4.4.1.2 Read Instruction 75
4.4.1.3 Associative-Search Instructions 77
4.4.2 Phrase Operation Mode 79
4.5 Comparison of the CAM to Other Similar Designs 81
4.6 Summary and Conclusion 81

V. Bit-Serial Word-Parallel Associative Memory 83
5.1 Introduction 83
5.2 BSWPAM Organization and General Operation Description 83
5.3 BSWPAM-Architecture Description 85
5.3.1 Data-Write Circuit Description 87
5.3.2 Address-Driver Description 90
5.3.3 Address-Decode Circuit Description 92
5.3.4 Bit-Serial Array Description 96
5.3.5 Data-Read Circuit and Slice Data-Read Circuit Descriptions .. 99
5.3.6 PE-Array Description 102
5.3.7 Address-Encoder Description 104
5.4 Bit-Serial Word-Parallel Associative-Memory Operation Description .. 106
5.4.1 Word Mode 107
5.4.1.1 Write and Read Instructions 107
5.4.1.2 Associative-Search Instructions 112
5.4.2 Phrase Mode 116
5.5 Operation Execution-Time Results 117
5.6 Summary and Conclusion 121

VI. Word-Organized Word-Parallel Extreme-Search Associative Memory 123

6.1 Introduction 123

6.2 ESAM Organization and General Operation Description 124

6.3 ESAM Architecture Description 124

6.3.1 Data-Write Circuit Description 125

6.3.2 Extreme-Array Description 126

6.3.3 Data-Read Circuit Description 127

6.3.4 Word-Select Circuit Description 129

6.4 ESAM Operation Description 131

6.4.1 Write Instructions 132

6.4.2 Read Instruction 133

6.4.3 Associative-Search Instructions 134

6.5 Comparison of ESAM to the ACAM 139

6.6 Summary and Conclusion 141

VII. Execution-Time and Layout-Dimension Analysis and Results 142

7.1 Introduction 142

7.2 Associative-Memory Execution-Time Analysis 142

7.2.1 Write and Read-Instruction Comparison 142

7.2.2 Associative-Search Comparison 145

7.2.2.1 Comparison of the CAM Architecture to the Single BSWPAM Architecture in the Execution of BPI, RCI Operations. 148

7.2.2.2 Comparison of the CAM Architecture to the Two BSWPAM Architecture in the Execution of BPI, RCI Operations. 149

7.2.2.3 Comparison of the CAM Architecture to the Four BSWPAM Architecture in the Execution of BPI, RCI Operations. 149

7.2.2.4 Comparison of the CAM Architecture to the ESAM Architecture in the Execution of BPI, RCI Operations. 152

7.2.2.5 Comparison of the Single BSWPAM Architecture to the CAM Architecture in the Execution of BPD, RCI Operations 153

7.2.2.6 Comparison of the Single BSWPAM Architecture to the ESAM Architecture in the Execution of BPI, RCI Operations 154

7.2.2.7 Comparison of the ESAM Architecture to the CAM Architecture in the Execution of BPD, RCD Operations 156

7.2.2.8 Comparison of the ESAM Architecture to the Single BSWPAM Architecture in the Execution of BPD, RCD Operations 158

7.2.2.9 Comparison of the CAM to the Single BSWPAM Architecture in the Word-Mode Execution of a 50% BPI, RCI and 50% BPD, RCI Instruction Mix. 159

7.2.2.10 Discussion 160

7.3 Memory-Layout Dimension Analysis 162

7.3.1 SRAM-Layout Dimensions 163

7.3.2 CAM-Layout Dimensions 164

7.3.3 BSWPAM-Layout Dimensions 165

7.3.4 ESAM-Layout Dimensions 166

7.4 Execution-Time Versus Layout-Dimensions Analysis 167

7.5 Summary and Conclusion 172

VIII. Conclusion 174

Appendix A: The Average Number of BPI, RCI Instructions a CAM Requires to Execute a BPD, RCI Operation 178

Appendices B-G: VHDL Code 182

References 183

Vita. 185

List of Figures

Figure		Page
1.	General Description of an Associative Memory Organization.	3
2.	Abstraction Levels in Digital Systems.	6
3.	SRAM Architecture.	29
4.	Write-Circuit Schematic.	32
5.	Driver Schematic.	37
6.	Decode-Circuit Schematic for the Thirty-Second Word in Memory.	40
7.	SRAM Array.	42
8.	SRAM-Cell Schematic.	43
9.	Read-Circuit Schematic.	44
10.	SRAM Timing Diagram.	47
11.	The CAM Architecture.	53
12.	CAM Write-Circuit Schematic.	55
13.	CAM Array.	60
14.	CAM-Cell Schematic.	61
15.	CAM Control-1 Driver Schematic.	64
16.	CAM Control-2 Driver Schematic.	65
17.	Word Circuit and Word/Search Line Driver Schematic.	68
18.	Match-Circuit Schematic.	73
19.	Multiple-Match-Resolution Circuit Schematic.	74
20.	Bit-Serial Word-Parallel Associative-Memory Organization.	84
21.	Single Bit-Serial Word-Parallel Associative-Memory Architecture.	86
22.	Bit-Serial Word-Parallel Associative-Memory Architecture Write-Circuit Schematic	88

23. Bit-Serial Array Associated with the Single Bit-Serial Word-Parallel Associative-Memory Architecture. 100

24. Bit-Serial Array Associated with the Two Bit-Serial Word-Parallel Associative-Memory Architecture. 101

25. Bit-Serial Array Associated with the Four Bit-Serial Word-Parallel Associative-Memory Architecture. 102

26. Bit-Serial Cell Schematic. 103

27. Single Bit-Serial Word-Parallel Associative-Memory PE Schematic. 108

28. Two Bit-Serial Word-Parallel Associative-Memory PE Schematic. 109

29. Four Bit-Serial Word-Parallel Associative-Memory PE Schematic. 110

30. Single Processing Element Address Encoder for the 128th Word of a 128 Word Memory. 113

31. The Extreme Array. The word-select data, and $\overline{\text{data}}$ port are not shown. . . 129

32. The ESAM Cell Memory Schematic 130

33. The ESAM Cell Logic Schematic 131

34. Comparison of the Write Times for a 32-Bit Data Field 144

35. Comparison of the Read Delays for a 32-Bit Data Field. 145

36. Comparison of the CAM to the Single BSWPAM Architecture in the Execution of BPI, RCI Operations. 149

37. Comparison of the CAM to the Two BSWPAM Architecture in the Execution of BPI, RCI Operations. 151

38. Comparison of the CAM to the Four BSWPAM Architecture in the Execution of BPI, RCI Operations. 153

39. Comparison of the Single BSWPAM Architecture to the CAM Architecture in the Execution of BPD, RCI Operations. 156

40. Comparison of the Single BSWPAM Architecture to the ESAM Architecture in the Execution of BPD, RCI Operation. 157

41. Comparison of the ESAM Architecture to the CAM Architecture in the Execution of BPD, RCD Operations. 158

42. Comparison of the ESAM Architecture to the Single BSWPAM Architecture in the Execution of BPD, RCD Operations. 160

43. Comparison of the CAM Architecture to the Single BSWPAM Architecture in the Word Mode Execution of a 50% BPI, 50% BPD, RCI Operation Mix . 161

44. Read-Instruction Execution Time Versus the Memory Layout Dimensions Per Bit. 168

45. Write-Instruction Execution Time Versus the Memory Layout Dimensions Per Bit 169

46. BPI, RCI Operation Execution Time Versus the Memory Layout Dimensions Per Bit. 170

47. BPD, RCI Operation Execution Time Versus the Memory Layout Dimensions Per Bit. 171

48. BPD, RCD Operation Execution Time Versus the Memory Layout Dimensions Per Bit. 172

List of Tables

Table		Page
1.	The Time to Execute BPI and BPD, RCI Operations in the Word or Phrase Mode for Each BSWPAM Architecture	22
2.	The Time to Execute BPD, RCD Operations in the Word or Phrase Mode for Each BSWPAM Architecture	23
3.	SRAM Layout Dimensions	33
4.	Write Circuit Transistor-Gate Dimensions for the SRAM Architecture.	34
5.	Data, $\overline{\text{Data}}$, and Data Precharge Delays for the SRAM Architecture.	35
6.	Least-Square Analysis for the SRAM Architecture.	36
7.	Driver Transistor-Gate Dimensions.	38
8.	Address, $\overline{\text{Address}}$, and Address Decode Delays for the SRAM Architecture.	39
9.	Decode Circuit Transistor-Gate Dimensions.	41
10.	Word-Select Delays for the SRAM Architecture.	41
11.	Read Circuit Transistor-Gate Dimensions.	45
12.	Read Delays for the SRAM Architecture.	46
13.	SRAM Write Access Time.	48
14.	SRAM Read Access Time.	49
15.	CAM Layout Dimensions.	56
16.	Write Circuit Transistor-Gate Dimensions for the CAM Architecture.	57
17.	Data, $\overline{\text{Data}}$, Mask, and Data Precharge Delays for the CAM Architecture.	58
18.	Least-Square Analysis for the CAM Architecture.	59
19.	CAM Cell Transistor-Gate Dimensions.	61
20.	Read Delays for the CAM Architecture.	62
21.	Control-1 Driver Transistor-Gate Dimensions.	64
22.	Control-2 Driver Transistor-Gate Dimensions.	66

23. Control, $\overline{\text{Control}}$, and Control(4) Delays for the CAM Architecture. 67

24. Control(3) and Control(4) Port Function. 69

25. Word-Select Delays for the CAM Architecture. 70

26. Search-Select Delays for the CAM Architecture. 71

27. Word-Match Delays for the CAM Architecture. 72

28. Write-All Time for the CAM Architecture. 76

29. Read Time for the CAM Architecture. 78

30. Search Time for the CAM Architecture. 80

31. Bit-Serial Word-Parallel Associative-Memory Component-Layout Dimensions. 89

32. Data, $\overline{\text{Data}}$, and Data Pre-Charge Delays for the Single Bit-Serial Word-Parallel Associative-Memory Architecture. 90

33. Data, $\overline{\text{Data}}$, and Data Precharge Delays for the Two Bit-Serial Word-Parallel Associative-Memory Architecture. 91

34. Data, $\overline{\text{Data}}$, and Data Pre-Charge Delays for the Four Bit-Serial Word-Parallel Associative-Memory Architecture 92

35. Least-Square Analysis for each Bit-Serial Word-Parallel Associative-Memory Architecture 93

36. Address, $\overline{\text{Address}}$, and Address Decode Delays for the Single Bit-Serial Word-Parallel Associative-Memory Architecture. 94

37. Address, $\overline{\text{Address}}$, and Address Decode Delays for the Two Bit-Serial Word-Parallel Associative-Memory Architecture. 95

38. Address, $\overline{\text{Address}}$, and Address Decode Delays for the Four Bit-Serial Word-Parallel Associative-Memory Architecture. 96

39. Slice Address, Slice $\overline{\text{Address}}$, and Slice Address Decode Delays for each BSWPAM Architecture. 97

40. Word-Select Delays for each Bit-Serial Word-Parallel Associative-Memory Architecture. 98

41. Bit-Slice Select Delays for each Bit-Serial Word-Parallel Associative-Memory Architecture. 99

42. Read Delays for the Bit-Serial Word-Parallel Associative-Memory Architectures 104

43. Slice Read Delays for the Single Bit-Serial Word-Parallel Associative-Memory. 105

44. Slice Read Delays for the Two Bit-Serial Word-Parallel Associative-Memory. 106

45. Slice Read Delays for the Four Bit-Serial Word-Parallel Associative-Memory. 107

46. PE Array *Control* Port Functions. 111

47. Bit-Serial Word-Parallel Associative-Memory Processing-Element Search Delays. 112

48. Encoder Transistor-Gate Dimensions. 114

49. Encoder Delays. 114

50. Write Time for the Bit-Serial Word-Parallel Associative-Memory Architectures. 115

51. Read Time for the Bit-Serial Word-Parallel Associative-Memory Architectures. 115

52. Bit-Serial Word-Parallel Associative-Memory Slice Read and Store Times. . . . 119

53. Bit-Serial Word-Parallel Associative-Memory Processing-Element Array Search Delays. 120

54. Extreme-Search Associative-Memory Layout Dimensions. 126

55. Data, $\overline{\text{Data}}$, and Data Precharge Delays for the ESAM Architecture. 127

56. Least-Square Analysis for the ESAM Architecture. 128

57. ESAM-Cell Transistor-Gate Dimensions. 130

58. Read Delays for the ESAM Architecture. 132

59. Control Delays for the ESAM Architecture. 133

60. Word-Select Delays for the ESAM Architecture. 134

61. Search-Select Delays for the ESAM Architecture. 135

62. Bit-Slice Evaluation Delays for Equivalence, Minima, and Maxima Operations... 136

63. Write-All Time for the ESAM Architecture. 137

64. Read Time for the ESAM Architecture. 138

65. $OP_{=}(C(s), db(s))$ Execution Time. 139

66. $OP_{min}(db(s))$ Execution Time. 140

67. $OP_{max}(db(s))$ Execution Time.. 141

ABSTRACT

The dissertation develops a decision criteria to select an associative-memory organization that minimizes the execution time of a mix of associative-search operations, and a decision criteria to estimate the layout dimensions of each organization for a specified memory size. The dissertation reclassifies Feng's associative-search operations into three hardware-influenced categories: bit-position independent (BPI), record-content independent (RCI); bit-position dependent (BPD), RCI; and BPD, record-content dependent (RCD). It develops a relationship between the categories and three associative-memory organizations: the CAM, the bit-serial word-parallel associative memory (BSWPAM), and the extreme-search associative memory (ESAM). A version of the CAM, three versions of the BSWPAM, and a version of the ESAM organizations were designed and simulated to show that for most memory sizes, BPI, RCI operations require less time when executed on a CAM, BPD, RCI operations require less time when executed on a BSWPAM organization, and for many memory sizes, BPD, RCD operations require less time when executed on an ESAM. The dissertation calculates the layout dimensions of each memory. The results indicate the CAM is the most area efficient followed by the single, and two BSWPAM, the ESAM, and the four BSWPAM.

A DECISION CRITERIA TO SELECT AN ASSOCIATIVE-MEMORY ORGANIZATION THAT MINIMIZES THE EXECUTION TIME OF A MIX OF ASSOCIATIVE-SEARCH OPERATIONS

I. Introduction

1.1 Introduction

The dissertation explains a research endeavor to compare several associative-memory organizations in the execution of associative-search operations. Chapter I presents key background information for the research endeavor, gives a concise purpose statement, and discusses the research approach. The chapter concludes with an overview of the remaining dissertation chapters.

1.2 Background

Computer performance is inversely related to the time required to execute a program [1]. Adding faster execution modes can improve performance, but only when they are used. For this reason, hardware designers study frequently used operations to determine whether they can be executed in a shorter time and within certain cost constraints with specialized hardware.

Equation (1) further clarifies the previous paragraph by showing that the overall speedup of adding specialized hardware equals the program execution time on the architecture without the specialized hardware divided by the program execution time on the enhanced architecture [1]. Furthermore, the overall speedup increases as the fraction of computation time that can take advantage of the specialized hardware and the speedup achieved by the enhancement increase.

$$Speedup_{overall} = \left\{ \frac{Execution\ time}{Execution\ time_{enhanced}} \right\} = \left\{ \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}} \right\} \quad (1)$$

Programs that perform associative-search operations on databases may experience an overall speedup on a computer enhanced with an associative memory. For example, artificial-intelligence programs require a large number and variety of associative-search operations that may be executed in a shorter time using an associative memory when compared to conventional memory [2, 3]. Therefore, the fraction of time the specialized hardware is used and the enhanced speedup may be high [4]. DeFiore estimates that the enhanced speedup potential of an associative memory is proportional to the $\log_2$ of the number of records when compared to sequential computers using a tree structure with an inverted list [5].

As shown in Figure 1, an associative-memory organization consists of a comparand register, a mask register, a memory array, a word/search-select register, a search-results register, and an output register of some form to execute the associative-search operations. The comparand register contains the data to be compared with the memory-array contents. The mask register contains the data used to identify the comparand-register bits that are compared with the memory-array contents. The word/search-select register performs a similar process as the mask register. It contains the data used to select the memory-array words involved in the write, associative-search, and read operations. The comparison results are stored in the search-results register. The contents of the search-results register can identify memory-array words to use in later operations, or identify a memory-array word to read to the output register.

Like a database, the memory array resembles a table. The rows of a database are records, and the columns are characters within the fields that comprise the records. As stated

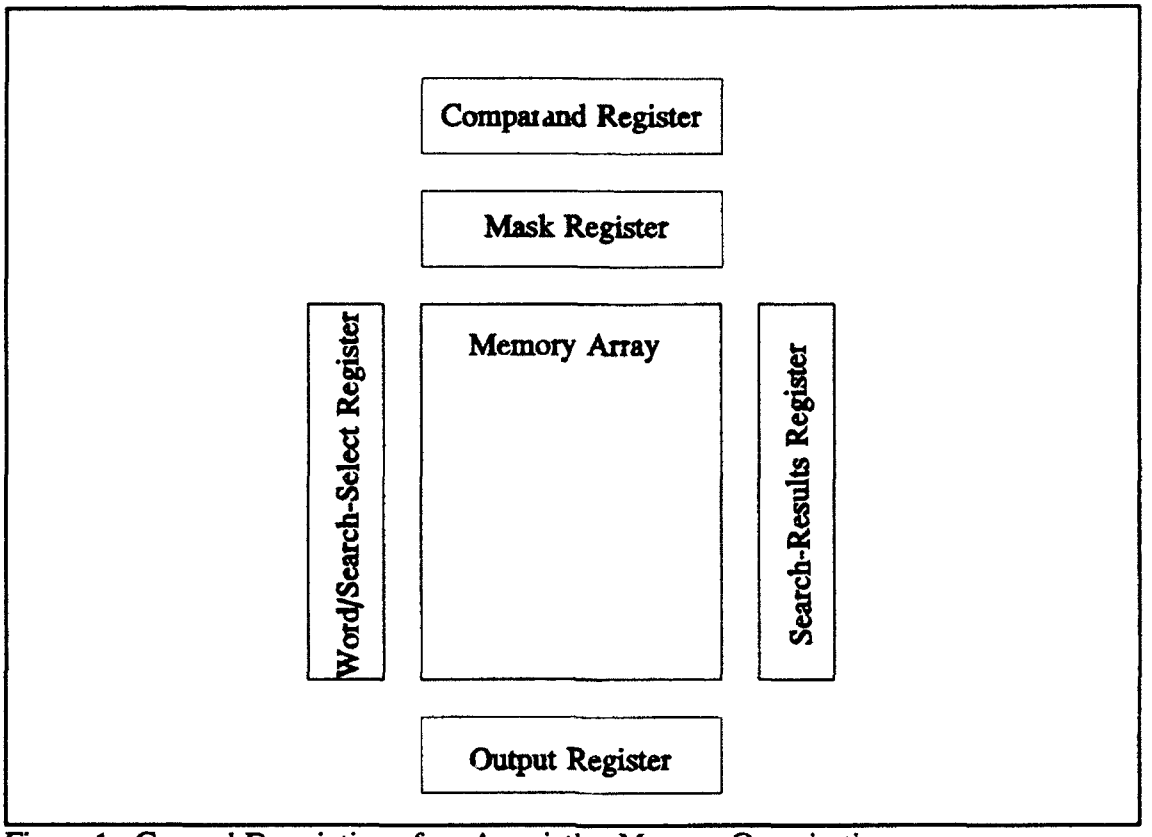

Figure 1. General Description of an Associative-Memory Organization.

in the previous paragraph, the rows of the memory array are words, while the columns are called bit-slices. Frequently, designers group consecutive bit-slices together to form fields, thus further fostering the analogy. The analogy does have a weakness though. Database records frequently cannot be represented by just one memory-array word. To compensate for this weakness, memory-array words can be grouped into units called phrases of sufficient length to adequately store the database record [3, 5].

If the number of phrases in the memory array equals or exceeds the number of records in the database, and the number of fields usable for data storage in the phrase equals or exceeds the number of fields in the database record, and the number of bits per field in the memory array is sufficient to represent the data within the database field, then the

database can be stored within the memory array. Once stored, the computer can execute associative-search operations on the database to find a requested record.

An associative memory can perform a variety of write, associative-search, and read operations, but the efficiency of the associative memory in executing these operations depends upon the associative-memory organization. Associative-memory organizations can be divided into many categories. Two of the most common organizations are: word-organized fully-parallel, and bit-serial word-parallel [3, 5, 6, 7, 8, 9]. These two and a third, as yet undefined, associative-memory organization will be examined throughout the research endeavor. The reason for selecting these organizations will be explained in Chapter II.

The best associative-memory organization to minimize the execution time of associative-search operations has yet to be determined. The algorithms used by each associative-memory organization give indications of the best organization selection for a particular associative-search operation, but that information alone does not reveal the enhanced speedup potential of a computer complemented with an associative memory. The time required to execute the associative-search operation algorithms for each organization is also needed. Approximately twenty years ago, Parhami described the need for this research, but the tools and technology were not available until recently [10].

1.3 Purpose

A variety of associative-memory organizations exist with each executing an associative-search operation differently. Likewise, a variety of associative-search operations exist to select for execution. The primary objective of this research endeavor is to develop a decision criteria to select an associative-memory organization that minimizes the execution time of a mix of associative-search operations. A secondary objective is to develop a rule-of-thumb criteria to estimate the memory size.

1.4 Approach

Selecting the best associative-memory organization to minimize the execution time of a mix of associative-search operations requires three analyses. The first analysis evaluates each associative-memory organization to determine the algorithms required to execute the associative-search operations. The analysis begins by identifying the pertinent associative-search operations. Each associative-search operation is expressed using mathematical operators executable by the associative-memory organizations. The analysis continues by explaining the algorithm steps each associative-memory architecture must execute to perform the associative-search operations, and it concludes by developing the equations to determine the execution time of a mix of associative-search operations for each associative-memory architecture. These equations are functions of the instruction mix, the number of bits in a field, and the time to execute each algorithm step.

The second analysis determines the time to execute each algorithm step. The architectural implementations of a variety of associative-memory organizations are presented, along with execution-speed simulation results. The next few paragraphs address design issues common to the architectures implemented for this research endeavor.

As shown in Figure 2, Armstrong identifies six levels of abstraction used in designing digital systems [11]. The highest level of abstraction is the processor-memory-switch, followed by chip, register, gate, circuit, and, at the lowest level, silicon. Using this hierarchy, Armstrong defines the term "design window," which is a range of levels over which the designer works. Armstrong states that a computer-system designer is concerned with the design window consisting of the processor-memory-switch level down to the gate-level, while a VLSI-chip designer is concerned with the design window consisting of the chip-level down to the silicon-level. This research endeavor uses the VLSI-chip design window.

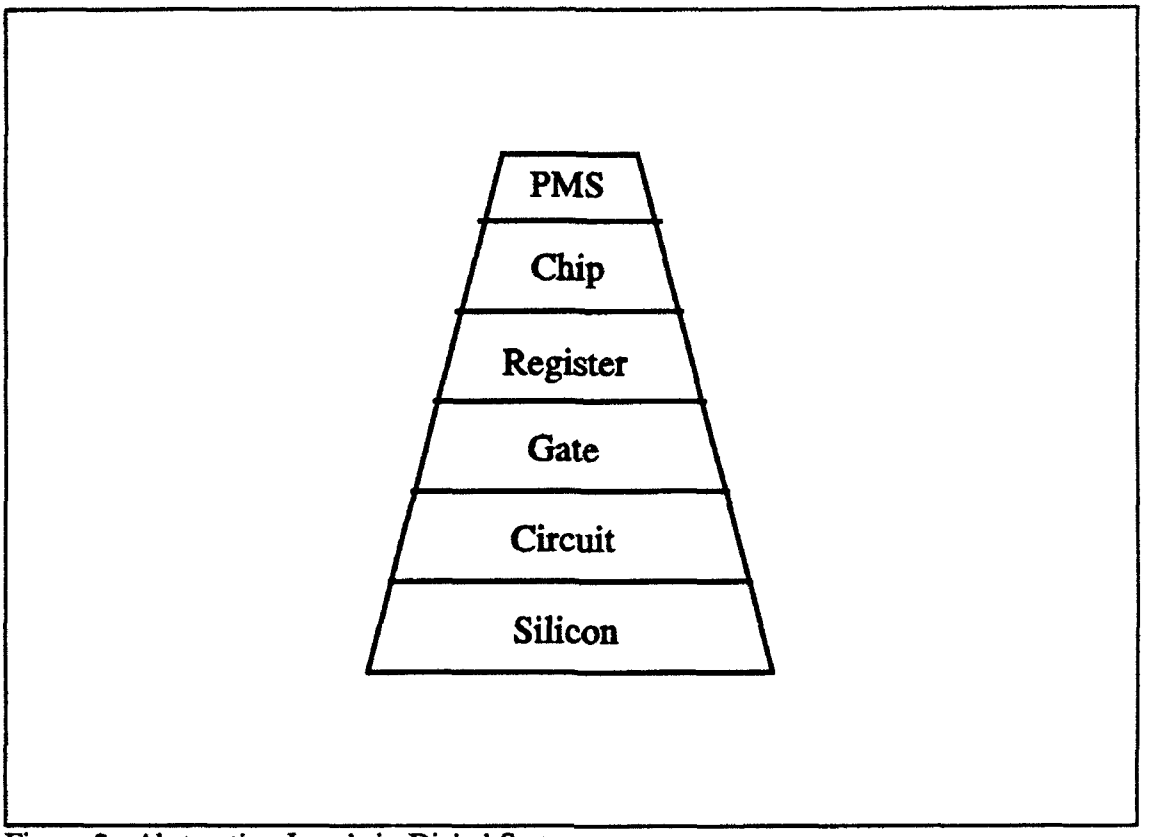

Figure 2. Abstraction Levels in Digital Systems.

The memory design approach is a mixture of top-down and bottom-up [11]. The top-down design approach identifies key components and defines their interconnection, while the bottom-up design approach takes advantage of existing designs to select the most appropriate circuit design for a component. The research endeavor uses very high speed integrated circuit hardware description language (VHDL) to verify that the memory design functions properly; uses Magic, with a 0.6 μm CMOS layout style, to create realistic circuits for simulation; and uses simulation program, integrated circuit emphasis (SPICE) to calculate algorithm step execution times.

In an effort not to skew the simulation results in favor of one associative-memory organization or another, gate-level designs are related between architectures. For example, each architecture uses a six-transistor static random-access memory cell of approximately the

same layout dimensions. Each design uses a common design approach in laying out the masks. Each architecture uses similar input stimuli and drivers during simulation, and so on.

SPICE is used to determine the algorithm-step execution times. Each architecture is simulated in the execution of its instruction set for a variety of memory-array sizes ranging from 8, 16, and 32-data-bit word lengths, and 4, 8, 16, 32, 64, and 128-word memory lengths. The critical-path delays for each instruction are measured and tabulated. The appropriate critical-path delays are summed and tabulated to estimate the approximate algorithm-step execution time for each memory-array size. From this tabulated data, the research endeavor develops characteristic equations approximating the algorithm-step execution times. The characteristic equations depend upon the number of bits in a word and the number of words in the memory array.

The third analysis combines the algorithm equations from the first analysis and the algorithm-step execution-time equations from the second analysis to complete the primary objective of the research endeavor. The results confirm a relationship between associative-search operations and the associative-memory organizations that affects the enhanced speedup as expressed in Equation (1).

Because the primary objective of the research endeavor requires the mask layout of each architecture, the mask-layout dimensions can be used to estimate the memory layout dimensions. This information can be used to accomplish the second objective of the research endeavor.

1.5 Overview

The dissertation is organized into eight chapters. Chapter II presents the theoretical foundation used throughout the research endeavor. It begins by presenting Feng's classification of associative-search operations. It then reorganizes Feng's classification into

three hardware-influenced associative-search operation categories and states a relationship between the hardware-influenced associative-search operation categories and the associative-memory organizations. Chapter III presents a static random-access memory used as a benchmark for the associative memories. The benchmark includes read and write execution-time, and memory-size information. Chapter IV presents the content-addressable memory. It presents both the organization and an architectural implementation. It also discusses the instruction set and the process to execute instructions. Chapter V presents the bit-serial word-parallel associative-memory organization and three architectural implementations: the single, two, and four bit-serial word-parallel associative-memory. Again the instruction set and the process to execute the instructions are discussed. Chapter VI discusses a new associative-memory organization designed to execute maxima and minima searches quickly, the extreme-search associative-memory organization. Chapter VII combines the information from Chapters II through VI to determine the execution time of a mix of associative-search operations. It also presents the memory-layout dimension analysis. Chapter VIII discusses the dissertation results and gives concluding remarks.

II. Theoretical Foundation Development

2.1 Introduction

Chapter II evaluates each associative-memory organization discussed in this research endeavor to determine the algorithms required to execute the associative-search operations. To accomplish this objective, the chapter begins by introducing and mathematically expressing Feng's classification of associative-search operations. Feng's classification is based on the arithmetic operators potentially executed upon a database. From the mathematical expressions, the chapter reveals a hardware-influenced reclassification of Feng's associative-search operations that gives insight into the relationship between the associative-memory organizations and the associative-search operations. The chapter concludes by developing mathematical expressions of the algorithms each associative-memory organization uses to execute the hardware-influenced associative-search operations.

2.2 Associative-Search Classification

Feng classified associative-search operations into the following categories: equivalence, threshold, between-limits, adjacency, extreme, and ordered-retrieval operations [12]. The equivalence operations can find all records equal-to the comparand, $Op_{=}$, or not-equal-to the comparand, $Op_{\neq}$. The threshold operations can find all records smaller-than the comparand, $Op_{<}$, greater-than the comparand, $Op_{>}$, not-smaller-than the comparand, $Op_{\geq}$, or not-greater-than the comparand, $Op_{\leq}$. The between-limits operations can find all records in a closed range, $Op_{\leq\leq}$, in an open range, $Op_{<<}$ or in a half open range, $Op_{\leq<}$, $Op_{<\leq}$. The adjacency operation can find the record that is nearest below the comparand, Op_{adj}. The extreme operations can find the maxima, the largest record among a set of records, Op_{max}, and the

minima, the smallest record among a set of records, Op_{min}. Order-retrieval operations can find all records in ascending order or descending order, Op_{asc}, Op_{des}.

These associative-search operations can be more exactly defined using a Boolean algebra. In an effort to build a logically correct and unambiguous set of associative-search operations, this section first defines the scalar Boolean operators, annotated by a superscript *s*, followed by the vector Boolean operators, annotated by a superscript ν, and functions.

2.2.1 The Boolean Algebra. Let $B=\{0, 1\}$. Define the scalar operator "and" by, $\cdot^s: B\times B\to B$ such that the relation is defined by,

$$\cdot^s=\{(0, 0, 0), (0, 1, 0), (1, 0, 0), (1, 1, 1)\} \tag{2}$$

Define the scalar operator "or" by, $+^s: B\times B\to B$ such that the relation is defined by,

$$+^s=\{(0, 0, 0), (0, 1, 1), (1, 0, 1), (1, 1, 1)\} \tag{3}$$

Define the scalar operator "complement" by, $^s B\to B$ such that the relation is defined by,

$$^s=\{(0, 1), (1, 0)\} \tag{4}$$

Define the scalar operator "equivalence" by, $\odot^s: B\times B\to B$ such that the relation is defined by,

$$\odot^s=\{(0, 0, 1), (0, 1, 0), (1, 0, 0), (1, 1, 1)\} \tag{5}$$

Define the scalar operator "exclusive-or" by, $\oplus^s: B\times B\to B$ such that the relation is defined by,

$$\oplus^s=\{(0, 0, 0), (0, 1, 1), (1, 0, 1), (1, 1, 0)\} \tag{6}$$

Given $n\in Z^+$ and $N\in N$, let B^N denote N ordered copies of B, that is, $B^N=B\times B\times ..\times B$; in particular,

$$B^N=\{(b_0, b_1, ..., b_n, ..., b_{N-1}) \mid 0\le n\le N-1;\ b_n\in B\forall n\} \tag{7}$$

Let $R\in N$ be the number of records within a database, and let $r\in Z^+$ be a specific record, where $0\le r\le R-1$. Let $S\in N$ be the number of fields within a record, and let $s\in Z^+$ be a specific

field, where $0 \leq s \leq S-1$. In general, S depends upon r, but the physical limitations of an associative-memory organization require that there exist a maximum value of S, $max(S)$. Once $max(S)$ is known, all the records within the database are redefined to contain $max(S)$ fields, and the dependence of S upon r is removed.

Given any r and any s there exists $T=T(r, s) \in N$, where T is the number of bits within a field. The value of T depends upon r and s, but again, the physical limitations of an associative-memory organization imply that there exists a maximum value for $T(r, s)$ for field s within the database, $max(T(s))$. Once $max(T(s))$ is used to define the number of bits in field s for all the records within the database, then the dependence of T upon r is removed and $T(r, s)$ is redefined as $T(s)$.

The next five equations define key components necessary for database searches. A field within a record is defined by,

$$field(r, s)=(bit(r, s, 0), ..., bit(r, s, t), ..., bit(r, s, T(s)-1)) \in B^{T(s)} \tag{8}$$

A record within the database is defined by,

$$record(r)=(field(r, 0), field(r, 1), ..., field(r, s), ..., field(r, S-1) \in B^{T(0)+T(1)+...+T(S-1)} \tag{9}$$

A database is defined by,

$$database=(record(0), record(1), ..., record(r), ..., record(R-1)) \in B^{R(T(0)+T(1)+...+T(S-1))} \tag{10}$$

A slice of the database consisting of $field(r, s)$ for all r is defined by,

$$db(s)=(field(0, s), field(1, s), ..., field(r, s), ..., field(R-1, s)) \in B^{R \cdot T(s)} \tag{11}$$

A field within a comparand, C, is defined by,

$$C(s)=(c(s, 0), ..., c(s, t), ..., c(s, T(s)-1))\in B^{T(s)} \quad (12)$$

The next five equations define vector operators and their equivalent execution with scalar operators. The equations compare each bit of field s within the comparand to each bit of field s and record r within the database. The operators are not overloaded to reduce ambiguity. Define the vector operator "and" by, $\cdot^{v}$: $B^{T(s)}\times B^{T(s)}\rightarrow B^{T(s)}$ such that,

$$C(s)\cdot^{v}field(r, s)=(c(s, 0)\cdot^{s}bit(r, s, 0), c(s, 1)\cdot^{s}bit(r, s, 1), ..., c(s, T(s)-1)\cdot^{s}bit(r, s, T(s)-1)) \quad (13)$$

Define the vector operator "or" by, $+^{v}$: $B^{T(s)}\times B^{T(s)}\rightarrow B^{T(s)}$ such that,

$$C(s)+^{v}field(r, s)=(c(s, 0)+^{s}bit(r, s, 0), c(s, 1)+^{s}bit(r, s, 1), ..., c(s, T(s)-1)+^{s}bit(r, s, T(s)-1)) \quad (14)$$

Define the vector operator "complement" by, ^{v}: $B^{T(s)}\rightarrow B^{T(s)}$ such that,

$$field(r, s)^{v}=(bit(r, s, 0)^{s}, bit(r, s, 1)^{s}, ..., bit(r, s, T(s)-1)^{s}) \quad (15)$$

Define the vector operator "equivalence" by, $\odot^{v}$: $B^{T(s)}\times B^{T(s)}\rightarrow B^{T(s)}$ such that,

$$C(s)\odot^{v}field(r, s)=(c(s, 0)\odot^{s}bit(r, s, 0), c(s, 1)\odot^{s}bit(r, s, 1), ..., c(s, T(s)-1)\odot^{s}bit(r, s, T(s)-1)) \quad (16)$$

Define the vector operator "exclusive-or" by, $\oplus^{v}$: $B^{T(s)}\times B^{T(s)}\rightarrow B^{T(s)}$ such that,

$$C(s)\oplus^{v}field(r, s)=(c(s, 0)\oplus^{s}bit(r, s, 0), c(s, 1)\oplus^{s}bit(r, s, 1), ..., c(s, T(s)-1)\oplus^{s}bit(r, s, T(s)-1)) \quad (17)$$

Equation (18) defines the scalar multiplication of a vector. Given $x\in B$, define the scalar multiplication of a vector by $\cdot^{v}$: $B\times B^{T(s)}\rightarrow B^{T(s)}$ such that,

$$x\cdot^{v}field(r, s)=(x\cdot^{s}bit(r, s, 0), x\cdot^{s}bit(r, s, 1), ..., x\cdot^{s}bit(r, s, T(s)-1)) \quad (18)$$

If $x=0$, then the result is the zero vector; otherwise, the result is *field(r, s)*.

The next two equations define operators that form scalars from vectors. A hardware analogy for Equation (19) is a wired-and operation. Likewise, a hardware analogy for

Equation (20) is a wired-or operation. Let $\alpha = (a_0, a_1, ..., a_r, ..., a_{R-1}) \in B^R$. Define $Prod(\alpha)$ by the following:

$$Prod(\alpha) = \prod_{r=0}^{R-1} a_r = a_0 {}^s a_1 {}^s, ..., {}^s a_{R-1} \tag{19}$$

Define $Sum(\alpha)$ by the following:

$$Sum(\alpha) = \sum_{r=0}^{R-1} a_r = a_0 +^s a_1 +^s, ..., +^s a_{R-1} \tag{20}$$

2.2.2 Feng's Associative-Search Operations. The components defined and operators derived in the previous section can be used to mathematically state Feng's associative-search operations. Several of the operations compare $C(s)$ with $db(s)$. The equality operation $Op_=: B^{T(s)} \times B^{R \cdot T(s)} \rightarrow B^R$ is defined by Equation (21).

$$\begin{aligned} Op_=(C(s), db(s)) &= (\prod_{t=0}^{T(s)-1} c(s, t) \odot^s bit(0, s, t), ..., \prod_{t=0}^{T(s)-1} c(s, t) \odot^s bit(R-1, s, t)) \\ &= (Prod(C(s) \odot^v field(0, s)), ..., Prod(C(s) \odot^v field(R-1, s))) \end{aligned} \tag{21}$$

In this equation, $C(s)$ is compared with $field(r, s)$ for all r to determine if they are equal element-by-element. When $C(s)$ equals $field(r, s)$, the resulting vector is T, and the product equals 1.

Similarly, $Op_{\neq}: B^{T(s)} \times B^{R \cdot T(s)} \rightarrow B^R$ is defined by Equation (22).

$$\begin{aligned} Op_{\neq}(C(s), db(s)) &= (\sum_{t=0}^{T(s)-1} c(s, t) \oplus^s bit(0, s, t), ..., \sum_{t=0}^{T(s)-1} c(s, t) \oplus^s bit(R-1, s, t)) \\ &= (Sum\ (C(s) \oplus^v field(0, s)), ..., Sum\ (C(s) \oplus^v field(R-1, s))) \end{aligned} \tag{22}$$

In this equation, when any element of the comparand does not equal the corresponding element in $field(r, s)$, the sum equals 1.

Equations (21) and (22) reveal two approaches to executing equivalence operations. The operations can be executed as a sequence of $T(s)$ bit-level comparisons or a single vector-level comparison. Though the two approaches yield the same result, the hardware implementation of each equation is different.

Next consider the threshold operations. To facilitate this discussion, define the binary vectors to use a magnitude number representation, where the most significant bit is the first element of the vector. In general, the result of a threshold operation between two fields is determined by the first non-matching pair of bits as the comparison progresses from the most significant to the least significant bit position. The vector, *S(C(s), field(r, s))* determines the first non-matching bits and is expressed by Equation (23).

$$S(C(s), field(r, s)) = (1, c(s, 0) \odot^{s} bit(r, s, 0), \ldots\ldots, \prod_{t=0}^{T(s)-2} c(s, t) \odot^{s} bit(r, s, t)) \qquad (23)$$

This equation establishes the significance of the bit position and its contents when executing threshold operations.

Equation (24) defines $OP_{<}$: $B^{T(s)} \times B^{R \cdot T(s)} \rightarrow B^{R}$.

$$\begin{aligned} Op_{<}(C(s), db(s)) = (&Sum(C(s) \cdot^{v} field^{v}(0, s) \cdot^{v} S(C(s), field(0, s))) \\ &\ldots\ldots, \\ &Sum(C(s) \cdot^{v} field^{v}(R-1, s) \cdot^{v} S(C(s), field(R-1, s)))) \end{aligned} \qquad (24)$$

If $C(s, 0)=1$ and $bit(r, s, 0)=0$, then the operation yields a 1. If $C(s, 0)=0$ and $bit(r, s, 0)=1$, then the operation yields a 0, but if $C(s, 0)$ equals $bit(r, s, 0)$, then $C(s, 1)$ and $bit(r, s, 1)$ are compared to determine whether the operation yields a result or $C(s, 2)$ and $bit(r, s, 2)$ should be compared. This process continues until every bit of a field is evaluated.

Equation (25) defines $OP_{>}$: $B^{T(s)} \times B^{R \cdot T(s)} \rightarrow B^{R}$.

$$\begin{aligned} Op_{>}(C(s), db(s)) = (&Sum(C(s)^{\nu} \cdot^{v} field(0, s) \cdot^{v} S(C(s), field(0, s))) \\ &\ldots\ldots, \\ &Sum(C(s)^{\nu} \cdot^{v} field(R-1, s) \cdot^{v} S(C(s), field(R-1, s)))) \end{aligned} \qquad (25)$$

If $C(s, 0)=0$ and $bit(r, s, 0)=1$, then the operation yields a 1. If $C(s, 0)=1$ and $bit(r, s, 0)=0$, then the operation yields a 0, but if $C(s, 0)$ equals $bit(r, s, 0)$, then $C(s, 1)$ and $bit(r, s, 1)$ are compared to determine whether the operation yields a result or $C(s, 2)$ and $bit(r, s, 2)$ should be compared. Likewise, this process continues until every bit of a field is evaluated.

Equation (26) defines $OP_{\geq}$: $B^{T(s)} \times B^{R \cdot T(s)} \rightarrow B^{R}$ as the complement of $Op_{<}$.

$$Op_{\geq}(C(s), db(s)) = Op_{<}(C(s), db(s))^{\nu} \tag{26}$$

Equation (27) defines $OP_{\leq}$: $B^{T(s)} \times B^{R \cdot T(s)} \rightarrow B^{R}$ as the complement of $Op_{>}$.

$$Op_{\leq}(C(s), db(s)) = Op_{>}(C(s), db(s))^{\nu} \tag{27}$$

The between-limits operations are defined using two threshold operations. The value, $u \in Z^{+}$, is the upper-limit field of the database record, where $0 \leq u \leq S-1$. The value, $l \in Z^{+}$, is the lower-limit field of the database record, where $0 \leq l \leq S-1$, $u \neq v$, $T(s)=T(u)=T(v)$. Equation (28) defines $OP_{<<}$: $B^{T(s)} \times B^{R \cdot T(s)} \rightarrow B^{R}$.

$$Op_{<<}(C(s), db(u), db(l)) = \left(Op_{<}(C(s), db(u)) \cdot^{\nu} Op_{>}(C(s), db(l))\right) \tag{28}$$

Equation (29) defines $OP_{\leq<}$: $B^{T(s)} \times B^{R \cdot T(s)} \rightarrow B^{R}$.

$$Op_{\leq<}(C(s), db(u), db(l)) = \left(Op_{<}(C(s), db(u)) \cdot^{\nu} Op_{\geq}(C(s), db(l)\right) \tag{29}$$

Equation (30) defines $OP_{<\leq}$: $B^{T(s)} \times B^{R \cdot T(s)} \rightarrow B^{R}$.

$$Op_{<\leq}(C(s), db(u), db(l)) = \left(Op_{\leq}(C(s), db(u)) \cdot^{\nu} Op_{>}(C(s), db(l))\right) \tag{30}$$

Equation (31) defines $OP_{\leq\leq}$: $B^{T(s)} \times B^{R \cdot T(s)} \rightarrow B^{R}$.

$$Op_{\leq\leq}(C(s), db(u), db(l)) = \left(Op_{\leq}(C(s), db(u)) \cdot^{\nu} Op_{\geq}(C(s), db(l))\right) \tag{31}$$

Equation (32) defines the maxima operation, Op_{max}: $B^{R \cdot T(s)} \rightarrow B^{R}$.

$$\begin{aligned} Op_{max}(db(s)) = (&Prod(Op_{\leq}(field(0, s), db(s))), \\ &Prod(Op_{\leq}(field(1, s), db(s))), \\ &\ldots\ldots, \\ &Prod(Op_{\leq}(field((R-1), s), db(s)))) \end{aligned} \tag{32}$$

Equation (33) defines the minima operation, Op_{min}: $B^{R \cdot T(s)} \rightarrow B^{R}$.

$$Op_{min}(db(s)) = (Prod(Op_{\geq}(field(0, s), db(s))), Prod(Op_{\geq}(field(1, s), db(s))), \ldots, Prod(Op_{\geq}(field(R-1, s), db(s))) \quad (33)$$

For Op_{max} and Op_{min}, the result of an operation performed upon one word in the memory array affects the result of the same operation performed upon other words in the memory array.

Adjacency and ordered-retrieval operations are executed as combinations of the above associative-search operations. For this reason, the research endeavor excludes the adjacency and ordered-retrieval operations from further discussion.

2.3 Reclassification of Feng's Associative-Search Operations

The mathematical expressions of Feng's associative-search operations reveal a new way of classifying the operations. The reclassification consists of three hardware-influenced associative-search operation categories. These categories are bit-position independent (BPI), record-content independent (RCI); bit-position dependent (BPD), record-content independent; and bit-position dependent, record-content dependent (RCD). Equivalence operations are BPI, RCI. Hardware can execute equivalence operations on a field within a record by comparing each bit of the comparand to the corresponding bit of the record field. Since the bit position does not affect the result, the operation is bit-position independent. Also since the result of searching one record does not depend upon the result of searching another record, the operation is record-content independent.

Threshold and between-limits operations are BPD, RCI. For hardware to determine whether a field within a comparand is greater than or less than a field within a record, some form of binary representation must be adapted, whether magnitude, sign-magnitude, two's complement, or most any other form. As revealed from the vector *S(C(s), field(r, s))* in

Equation (23), the bit position and content within the field are used to represent the value of the number, so these operations are bit-position dependent. Like the equivalence operations, the result of searching one record does not depend upon the result of searching another record, so the operation is record-content independent.

Extreme operations are BPD, RCD. Since numbers are represented in binary, bit position and content are used to determine the magnitude, implying bit-position dependence. Also, since the operation compares database records to each other, the result depends upon the contents of the database not just the records, implying record-content dependence.

Feng's associative-search operations have been reclassified by the way the data are searched and compared in an associative-memory organization. Next, this chapter states a relationship between the three hardware-influenced associative-search operations categories and the three associative-memory organizations selected for this research endeavor.

2.4 A Relationship Between the Hardware-Influenced Associative-Search Operation Categories and the Associative-Memory Organizations

A relationship between the hardware-influenced associative-search operation categories and the associative-memory organizations exists. Section 2.4.1 explains the relationship between the BPI, RCI operations and the word-organized fully-parallel associative-memory organization, also called the content-addressable memory (CAM) organization. Section 2.4.2 explains the relationship between the BPD, RCI operations and the bit-serial word-parallel associative-memory (BSWPAM) organization, and Section 2.4.3 explains the relationship between the BPD, RCD operations and the extreme-search associative-memory (ESAM) organization. This section also evaluates the algorithms used by each associative-memory organization to execute each hardware-influenced associative-search operation category.

2.4.1 CAM Description. BPI, RCI operations exhibit a granularity to the bit level. As is seen in Equations (21) and (22), the element *c(s, t)* can be simultaneously compared to *bit(r, s, t)* for all *r*, *s*, and *t*, followed by an operation to combine the bit-level comparison. Hardware of similar granularity requires a comparator for every bit of the memory array and a logical operator to combine the comparisons into a single result per word. The CAM has such an organization.

The CAM organization distributes to each bit within the memory array the computational logic necessary to perform an association instruction -- typically an equivalence association [13]. Data are typically stored within and retrieved from the memory array word-by-word without using an address. A comparand is simultaneously compared with the contents of the memory array. The comparison results of each unmasked bit within a word are combined into a single bit answer which is stored in the search-results register.

CAM architectures have been designed with a variety of capability levels. The simplest CAM will always compare every bit of every word. No mask register or word/search-select register is required, but its search capabilities are limited to identifying whether a datum is stored within the memory array. The mask register allows a portion of the comparand to be compared to a portion of the memory array to determine whether the exact datum or similar data are stored within the memory array. The addition of the word/search-select register allows a subset of words to be searched. With the addition of the mask and word/search-select register, greater-than and less-than operations can be executed using a process of elimination [14].

A certain portion of each word in the CAM is used for purposes other than data storage. Both word and phrase-operation modes require T_{wdf} bits to identify the word as valid or invalid. The phrase mode requires an additional T_{pf} bits to identify the word position within a phrase.

The relationship between the BPI, RCI operations and the CAM organization allows a CAM to search the entire database against the comparand and to form the result in the time to execute one BPI, RCI instruction, τ_{cam}. In the case where phrases are used, the BPI, RCI instructions can be executed in a time equal to the number of memory-array words required to form a phrase, *M*, multiplied by the associative-search instruction execution time, τ_{cam}, plus *(M-1)* transfers between words within the same phrase multiplied by the time required to transfer each search result from one word to another, τ_{Tcam}. When possible, the CAM word lengths should meet or exceed the record lengths to reduce the value of *M*.

The CAM cannot execute BPD operations as efficiently as BPI instructions. The CAM organization processes data with a finer grain than required to execute BPD operations. The CAM executes BPD operations using successive equivalence operations.

The average number of BPI, RCI instructions required to execute a BPD, RCI threshold operation is derived in Appendix A. If the number of bits in a field equals *T(s)*, then on average the CAM requires $((T(s)-1)/2)+(1/(2^{T(s)}))$ BPI, RCI instructions to execute a BPD, RCI threshold operation. Since τ_{cam} is the time to execute a BPI, RCI instruction, the average time to execute a threshold operation in the word mode is $((T(s)-1)/2 + (1/(2^{T(s)})))\tau_{cam}$, and if phrases are used, the average execution time is $M((T(s)-1)/2 + (1/(2^{T(s)})))\tau_{cam} + (M-1)\tau_{Tcam}$. BPD, RCI range operations require two threshold operations.

The CAM executes BPD, RCD extreme operations bit-serially [15]. To find the maxima, the process begins by comparing a 1 placed in the most significant bit position of the comparand to the most significant bit-slice of the database field. If any match occurs, then the first bit of the maxima is a 1; otherwise, it is a 0. Next a 1 is placed in the second most significant bit position of the comparand, and the two most significant bit-slices are compared. The result will indicate whether the second most significant bit-slice is a 1 or a 0. The

process continues until all $T(s)$ bits of the maxima are formed. The CAM requires $T(s)$ BPI, RCI instructions to form the maxima in the comparand, and if phrases are used, the algorithm requires $MT(s)$ BPI, RCI instructions. This implies that for the word mode, the time required to execute a maxima operation is $T(s)\,\tau_{cam}$, and for the phrase mode, $MT(s)\,\tau_{cam} + (M\text{-}1)\,\tau_{Tcam}$. The minima is formed in a similar fashion.

2.4.2 BSWPAM Organization Description. BPD, RCI operations exhibit a granularity at the field level with the result of searching one field not dependent upon the result of searching another field. Though $C(s)$ can be compared to *field(r, s)* for all r and a specific s simultaneously, the contents of the field are compared sequentially. Hardware of similar granularity requires a comparator for every word in the memory array. The BSWPAM has such an organization.

The BSWPAM organization distributes to each bit in the memory array the logic to execute write and read instructions and to each word in the memory array the logic to execute associative-search operations. In this research endeavor, the BSWPAM organization represents an entire class of associative-memory architectures. Each architecture compares a sequences of bits within the comparand to a sequence of bit-slices within the memory array and stores the results in the search-results register to be combined with later comparisons [16]. The major difference between each architecture is the number of bit-slices compared simultaneously. The single BSWPAM architecture compares a single bit of the comparand to a single bit-slice of the memory array at a time. The two BSWPAM architecture compares two bits of the comparand to two bit-slices of the memory array at a time, and the four BSWPAM architecture compares four bits of the comparand to four bit-slices of the memory array at a time. The eight BSWPAM architecture was considered, but not implemented because of the excessive wiring associated with reading eight bit-slices of the

memory array at a time. Like the CAM, words require T_{vdf} bits to identify the word as valid and T_{pf} bits to identify the word position in a phrase.

As the number of bits which are simultaneously compared increases, the number of comparisons decreases, but the amount of time required to perform the comparison may increase due to the increased complexity of associated logic circuitry. Increasing the time to execute the associative-search operation negates a portion of the speed enhancements found by reducing the number of comparisons. Furthermore, as the number of simultaneously compared bits increases, so does the relative size of logic and data transfer circuit. These issues will be discussed in detail in Chapter V.

All three BSWPAM architectures execute BPI, RCI equivalence instructions and BPD, RCI threshold instructions using the same algorithm. The single BSWPAM architecture compares a single bit of the comparand to a corresponding bit-slice of the memory array in a time, τ_{c1}. The two BSWPAM architecture compares two consecutive bits of the comparand to the two corresponding bit-slices of the memory array in a time, τ_{c2}, and the four BSWPAM architecture compares four consecutive bits of the comparand to the four corresponding bit-slices of the memory array in a time, τ_{c4}. The comparison results are stored and combined with other bit-slice comparisons to form a final result. Using $(M-1)\,\tau_{T1}$ as the number of transfers required in a phrase multiplied by the time to execute a transfer, Table 1 gives the time to execute a BPI, RCI or BPD, RCI instruction in the word or phrase mode.

Each BSWPAM architecture executes extreme operations differently. The single BSWPAM architecture begins a maxima operation by setting each bit of the comparand field involved in the comparison to 1. The most significant bit of the comparand and the corresponding memory array bit-slices are compared to determine the fields within the memory array that have the potential to be the maxima. This comparison result is used to identify the subset of words in the memory array to compare to the second most significant

Table 1

The Time to Execute BPI and BP~ RCI Operations
in the Word or Phrase Mode for Each BSWPAM Architecture.

	Word Mode	Phrase Mode
Single Bit-Serial Word-Parallel Associative Memory	$(T(s)+T_{wdf})t_{c1}$	$M(T(s)+T_{wdf}+T_{pf})t_{c1} + (M-1)t_{T1}$
Two Bit-Serial Word-Parallel Associative Memory	$(\lceil T(s)/2 \rceil+\lceil (T_{wdf})/2 \rceil)t_{C2}$	$M(\lceil T(s)/2 \rceil + \lceil (T_{wdf}+T_{pf})/2 \rceil)t_{C2} + (M-1)t_{T1}$
Four Bit-Serial Word-Parallel Associative Memory	$(\lceil T(s)/4 \rceil+\lceil (T_{wdf})/4 \rceil)t_{C4}$	$M(\lceil T(s)/4 \rceil + \lceil (T_{wdf}+T_{pf})/4 \rceil)t_{C4} + (M-1)t_{T1}$

bit-slice of the memory array. The algorithm continues until the least significant bit-slice of the memory array is compared.

The two BSWPAM architecture sets the comparand to a 01 sequence and executes a greater-than operation on the first two bits. If a match occurs, then the next evaluation should be 10, otherwise 00. If 10 is the choice and a match occurs, the two extreme bits are 11, otherwise they are 10. If 00 is the choice and a match occurs, then the two extreme bits are 01, otherwise they are 00. These two searches are executed for each two bit-slice comparison.

The four BSWPAM architecture sets the comparand to a 0111 sequence and executes a greater-than operation on the first four bits. If a match occurs, then the next evaluation should be 1011 otherwise 0011, etc. These four bit-slice evaluations are executed for each four bit-slice comparisons. Table 2 gives the time for each BSWPAM architecture to execute a BPD, RCD operation in the word or phrase mode.

2.4.3 ESAM Organization Description. BPD, RCD operations exhibit a granularity at the field level with the result of searching one field dependent upon the result of searching

Table 2

The Time to Execute BPD, RCD Operations in the Word or Phrase Mode for Each BSWPAM Architecture.

	Word Mode	Phrase Mode
Single Bit-Serial Word-Parallel Associative Memory	$(T(s)+T_{wdf})t_{c1}$	$M(T(s)+T_{wdf}+T_{pf})t_{c1} + (M-1)t_{T1}$
Two Bit-Serial Word-Parallel Associative Memory	$(2\lceil T(s)/2\rceil+\lceil(T_{wdf})/2\rceil)t_{C2}$	$M(2\lceil T(s)/2\rceil + \lceil(T_{wdf}+T_{pf})/2\rceil)t_{C2} + (M-1)t_{T1}$
Four Bit-Serial Word-Parallel Associative Memory	$(4\lceil T(s)/4\rceil+\lceil(T_{wdf})/4\rceil)t_{C4}$	$M(4\lceil T(s)/4\rceil + \lceil(T_{wdf}+T_{pf})/4\rceil)t_{C4} + (M-1)t_{T1}$

another field. The search results are not formed from comparing the database to a comparand, but rather from comparing the database to itself. Hardware of similar granularity requires a comparator used to compare fields against each other. The ESAM has such an organization.

The ESAM organization has characteristics that resemble both the CAM and the BSWPAM organizations. Like the CAM, each cell within the memory array can execute logical operations. The ESAM cell can compare its content to a comparand bit for an equivalence operation. It can also compare its content to other bits in the bit-slice to identify the words that may be either a maxima or a minima. Like the BSWPAM organization, the results of comparing one bit-slice of the memory array are sequentially combined with the comparison results of the next bit-slice of the memory array. This rippling effect is not as efficient as the CAM in executing equivalence operations, but is necessary to execute BPD operations.

Since each cell in a field can execute an equivalence operation, the ESAM organization can equivalence search the entire database against the comparand and form the

result in the time to execute one BPD, RCD instruction, τ_{ex}. In the case where phrases are used, the BPI, RCI operation can be executed in a time equal to the number of associative-memory words required to form a phrase, M, times the BPD, RCD instruction execution time, τ_{ex}, plus the time required to transfer each search result from one word to another, $(M-1)\tau_{Tex}$.

The ESAM executes BPD, RCI operations in the same manner as the CAM. Therefore, the average time to execute a threshold search is $((T(s)-1)/2 + (1/(2^{T(s)})))\tau_{ex}$, and if phrases are used, the average execution time is $M((T(s)-1)/2 + (1/(2^{T(s)})))\tau_{ex} + (M-1)\tau_{Tex}$. In the word mode, the ESAM organization executes a BPD, RCD instructions in a time, τ_{ex}, and in the phrase mode $M\tau_{ex} + (M-1)\tau_{Tex}$.

2.4.4 Associative-Memory Execution of a Mix of Associative-Search Operations. The previous section defined the amount of time each of five associative-memory architectures require to execute three different categories of associative-search instructions. This section will unify the algorithms to provide a single expression for the time to execute a mix of associative-search operations.

Let x be the percentage of associative-search operations that are BPI, RCI. Let y be the percentage of associative-search operations that are BPD, RCI. The value y is calculated with the assumption that range operations require two threshold operations. Let z be the percentage of associative-search operations that are BPD, RCD. This implies that $x+y+z=1$. Also, define $\tau_{word\ mode}$ to be the average word-mode execution time, and $\tau_{phrase\ mode}$ to be the average phrase-mode execution time.

In the word mode, the CAM execution time for a mix of associative-search operations is:

$$\tau_{word\ mode}=(x+(\frac{T(s)-1}{2}+\frac{1}{2^{T(s)}})y+T(s)z)\tau_{cam} \qquad (34)$$

Substituting $z=1-x-y$ and reorganizing yields:

$$\tau_{word\ mode}=((1-T(s))x-(\frac{T(s)+1}{2}-\frac{1}{2^{T(s)}})y+T(s))\tau_{cam} \quad (35)$$

The phrase mode requires M repetitions of Equation (35) plus the time to transfer the comparison results from one word to another within a phrase.

$$\tau_{phrase\ mode}=M((1-T(s))x-(\frac{T(s)+1}{2}-\frac{1}{2^{T(s)}})y+T(s))\tau_{cam}+(M-1)\tau_{Tcam} \quad (36)$$

The single BSWPAM word-mode execution time for a mix of associative-search operations is:

$$\tau_{word\ mode}=(T(s)+T_{wdf})\tau_{c1} \quad (37)$$

The phrase-mode execution time for a mix of associative-search operations is:

$$\tau_{phrase\ mode}=M(T(s)+T_{wdf}+T_{pf})\tau_{c1}+(M-1))\tau_{T1} \quad (38)$$

Tables 1 and 2 provide the operation execution-time equations used in Equations (37) and (38). Since this architecture uses algorithms requiring the same number of step for each associative-search operation category, the number of bit-slice comparisons required to execute an associative-search operation is constant. Tables 1 and 2 also provide the operation execution-time equations used in Equations (39) and (42).

The two BSWPAM word-mode execution time for a mix of associative-search operations is:

$$\tau_{word\ mode}=(\lceil\frac{T(s)}{2}\rceil+\lceil\frac{T_{wdf}}{2}\rceil+2\lceil\frac{T(s)}{2}\rceil)\tau_{c2} \quad (39)$$

The phrase-mode execution time for a mix of associative-search operations is:

$$\tau_{phrase\ mode}=M(\lceil\frac{T(s)}{2}\rceil+\lceil\frac{T_{wdf}+T_{pf}}{2}\rceil+2\lceil\frac{T(s)}{2}\rceil)\tau_{c2}+(M-1)\tau_{T1} \quad (40)$$

The four BSWPAM word-mode execution time for a mix of associative-search operations is:

$$\tau_{word\ mode}=(\lceil\frac{T(s)}{4}\rceil+\lceil\frac{T_{wtf}}{4}\rceil+3\lceil\frac{T(s)}{4}\rceil)\tau_{cd} \quad (41)$$

The phrase-mode execution time for a mix of associative-search operations is:

$$\tau_{phrase\ mode}=M(\lceil\frac{T(s)}{4}\rceil+\lceil\frac{T_{wtf}+T_{pf}}{4}\rceil+3\lceil\frac{T(s)}{4}\rceil)\tau_{cd}+(M-1)\tau_{TI} \quad (42)$$

Finally, the ESAM word-mode execution time for an associative-search operation mix is:

$$\tau_{word\ mode}=(x+(\frac{T(s)-1}{2}+\frac{1}{2^{T(s)}})y+z)\tau_{ex} \quad (43)$$

Once the substitution of $z=1-x-y$ is made and the reorganization is complete, the equation becomes:

$$\tau_{word\ mode}=(1+(\frac{T(s)-3}{2}+\frac{1}{2^{T(s)}})y)\tau_{ex} \quad (44)$$

The ESAM phrase-mode execution time for a mix of associative-search operations is:

$$\tau_{phrase\ mode}=M(1+(\frac{T(s)-3}{2}+\frac{1}{2^{T(s)}})y)\tau_{ex}+(M-1)\tau_{Tex} \quad (45)$$

The execution-time equations for each associative-memory architecture reveal strengths and weakness in the algorithms used to execute the associative-search instruction mix. For example, the CAM and ESAM architectures require fewer algorithm steps to execute BPI, RCI operations when compared to the three different BSWPAM architectures. The CAM and ESAM architectures require about one-half the algorithm steps to execute BPD, RCI operations when compared to the single BSWPAM architecture. The CAM, ESAM and two BSWPAM architectures require about the same number of algorithm steps to execute BPD, RCI operations. The four BSWPAM architecture requires about one-half the algorithm steps to execute BPD, RCI operations when compared to the CAM and ESAM

architectures. The ESAM architecture requires fewer algorithm steps to execute BPD, RCD operations when compared to the other architectures considered in this research endeavor.

Better algorithms do not necessarily lead to shorter execution times. The best architecture in executing a mix of associative-search operations cannot be determined until the values of τ_{cam}, τ_{c1}, τ_{c2}, τ_{c4}, τ_{ex}, τ_{T1}, and τ_{Tex} are known. These times depend upon the organization, the architecture that realizes the organization, and the fabrication technology used to implement the architecture. These values are not easily estimated, though they can be obtained through simulations and measurements.

2.5 Summary

Chapter II developed the theoretical foundation for the research endeavor. Section 2.2 introduced Feng's classification of associative-search operations, then continued to mathematically define Feng's associative-search operations with emphasis upon logical operators executable by an associative-memory organization. From these mathematical expressions, the chapter redefined Feng's associative-search operations into three hardware-influenced associative-search operation categories. This reclassification identified information used to develop mathematical expressions for the execution time of a mix of associative-search operations. This completes the first analysis which was introduced in Chapter I.

III. Static Random-Access Memory

3.1 Introduction

Though the research endeavor deals with associative-memory organizations, it also needs a random-access memory (RAM) to serve as a benchmark. Since the associative-memory architectures in this research are designed using static RAM (SRAM), an SRAM is a realistic benchmark for both the read and write access-time comparisons and the memory layout-dimension comparisons.

The chapter develops the characteristic equations to approximate the read and write-access times for a variety of memory-array sizes. To accomplish this objective, the chapter begins by describing the SRAM organization. The chapter continues by explaining the SRAM architecture from the chip level down to the transistor level. As each component is described, the measured critical-path delays are presented. Next, the chapter sums the critical-path delays to yield the SRAM read and write-access times for a variety of memory-array sizes. This tabulated data is used to develop the characteristic equations which approximate the read and write-access times.

For a complete VHDL description of the SRAM architecture, refer to Appendix B.

3.2 SRAM-Organization Description

The SRAM organization consists of an address register, an address-decode circuit, a data register, and a memory array, called the SRAM array. Central to the SRAM organization is a two-dimensional array of memory cells that form the SRAM array. These memory cells, called SRAM cells, are organized into words, which can be accessed using an address stored in the address register. The SRAM array consists of I words with each word containing J SRAM cells. The SRAM is designed under the assumption that a word is the

basic computer processing element rather than a subset of SRAM cells within a word, so the design allows the host to access words within the SRAM array but not individual SRAM cells within a word. The SRAM can execute two instructions, read and write.

3.3 SRAM-Architecture Description

Figure 3 shows the SRAM architecture. The SRAM consists of five components: the data-write circuit, the address driver, the address-decode circuit, the SRAM array, and the data-read circuit. In turn, these components are generated from repetitions of less complicated circuits. This section describes, to the transistor level, the components that form the SRAM architecture.

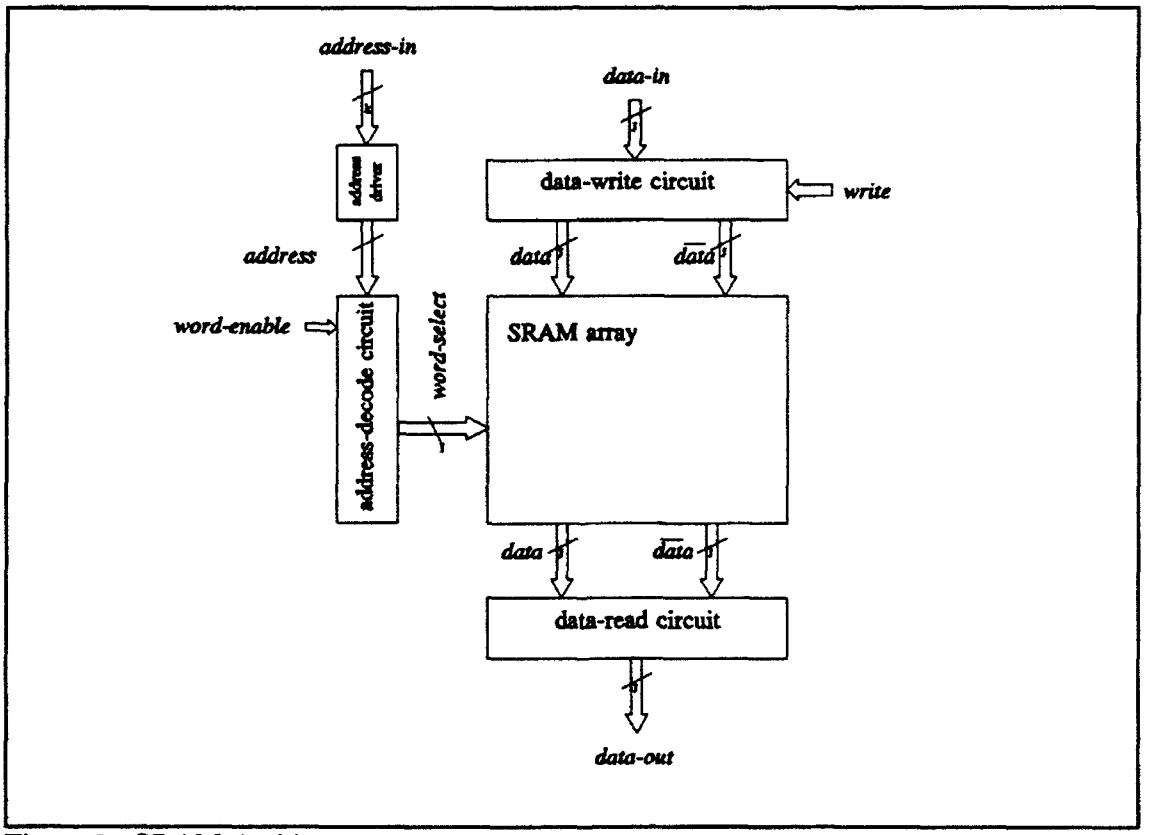

Figure 3. SRAM Architecture.

The SRAM architecture interfaces with the host using five ports: *data-in*, *data-out*, *address-in*, *write*, and *word-enable*. The host supplies data to the SRAM architecture using

the *J*-element *data-in* port and retrieves data using the *J*-element *data-out* port. It selects the location of data storage and retrieval using the *K*-element *address-in* port (where $K=\lceil log_2 J \rceil$) and controls the write and read processes using the *write* and *word-enable* ports.

The input ports are used as the references to measure the critical-path delays. Beginning in this chapter and continuing through Chapter VI, all critical-path delays are referenced from the 50% point of the input-port transition between ground and V_{dd}. The ports representing the critical-path delays are measured at the 10%, 50%, and 90% points of their transition to develop a course approximation of the slope, but the tabulated data presented within the chapters will provide only the 50% point measurements. Within the tabulated data, t_r, identifies the port propagation delay as it transitions from ground to V_{dd}, and t_f represents the port propagation delay as it transitions from V_{dd} to ground.

The transition slopes are examined to ensure that they are appropriate for the circuit design being stimulated. This requires a certain amount of engineering judgement. For example, CMOS gates driving relatively small loads transition within a couple of nanoseconds, but when driving larger loads will require more time depending upon the ability of the circuit to drive the load. Transitions that are in the critical path are optimized, frequently at the expense of less critical transitions.

The tabulated data accuracy depends in part upon the simulation tool accuracy. HSPICE accuracy tolerances are between 1% and 5% using the same transistor models, the same extractor, and similar circuit designs [17]. Measured results are rounded off to the nearest 0.1 ns.

Within this chapter, each circuit description will identify potential critical paths and give the measured propagation delays for word lengths of *J* equal to 8, 16, and 32 SRAM cells and SRAM array lengths of *I* equal to 4, 8, 16, 32, 64, and 128 words. Due to insufficient computer memory, the entire SRAM architecture could not be simulated for every SRAM-

array size. To allow the critical-path delay measurements, the SRAM array is reduced to a single bit-slice of *I-1* SRAM cells attached to a single *J*-SRAM cell word. The bit-slice of SRAM cells allows accurate modelling of SRAM array length dependent critical-path delays, while the word allows accurate modelling of word length dependent critical-path delays. In combination, they allow the measurement of delays dependent upon both *I* and *J*.

3.3.1 Data-Write Circuit Description. The first of the five SRAM components discussed is the data-write circuit. The data-write circuit accepts the *data-in* and *write* port stimuli from the host to form the *data* and $\overline{data}$ port stimuli. The *data-in*, *data*, and $\overline{data}$ ports consist of *J* elements, and an element within each port is identified as *data-in(j)*, *data(j)*, and $\overline{data(j)}$, respectively.

The data-write circuit consists of *J* write circuits. Figure 4 shows the write-circuit schematic for *data-in(j)* [13]. Table 3 gives its layout dimensions, and Table 4 gives the transistor-gate dimensions. Table 3 also gives the layout dimensions of the other circuits discussed within this chapter. The write circuit performs two functions. During a write instruction, the host places the data to be stored in the SRAM array onto *data-in* and enables the *write* port. The write circuit inverts the value on *data-in(j)* twice to form *data(j)* and three times to form $\overline{data(j)}$. The inverters are cascaded so that each progressive inverter has n-type and p-type transistor widths two times as large as the previous inverter. The cascading is necessary to drive the 11 femtofarad (fF) per word capacitance found on the *data(j)* and $\overline{data(j)}$ ports. *Data(j)* and $\overline{data(j)}$ can be either grounded or set to V_{dd}-V_{tn}, but not V_{dd} due to the in-line n-channel transistors (V_{tn} is the n-channel transistor threshold voltage). During a read instruction, *data(j)* and $\overline{data(j)}$ precharge to V_{dd}-V_{tn}.

The time to form the *data(j)* and $\overline{data(j)}$ ports has the potential to be a critical-path delay for both the read and write instructions. The data/$\overline{data}$ delay is referenced from the 50% point of the *write* port transition from ground to V_{dd} and is initiated from a precharged

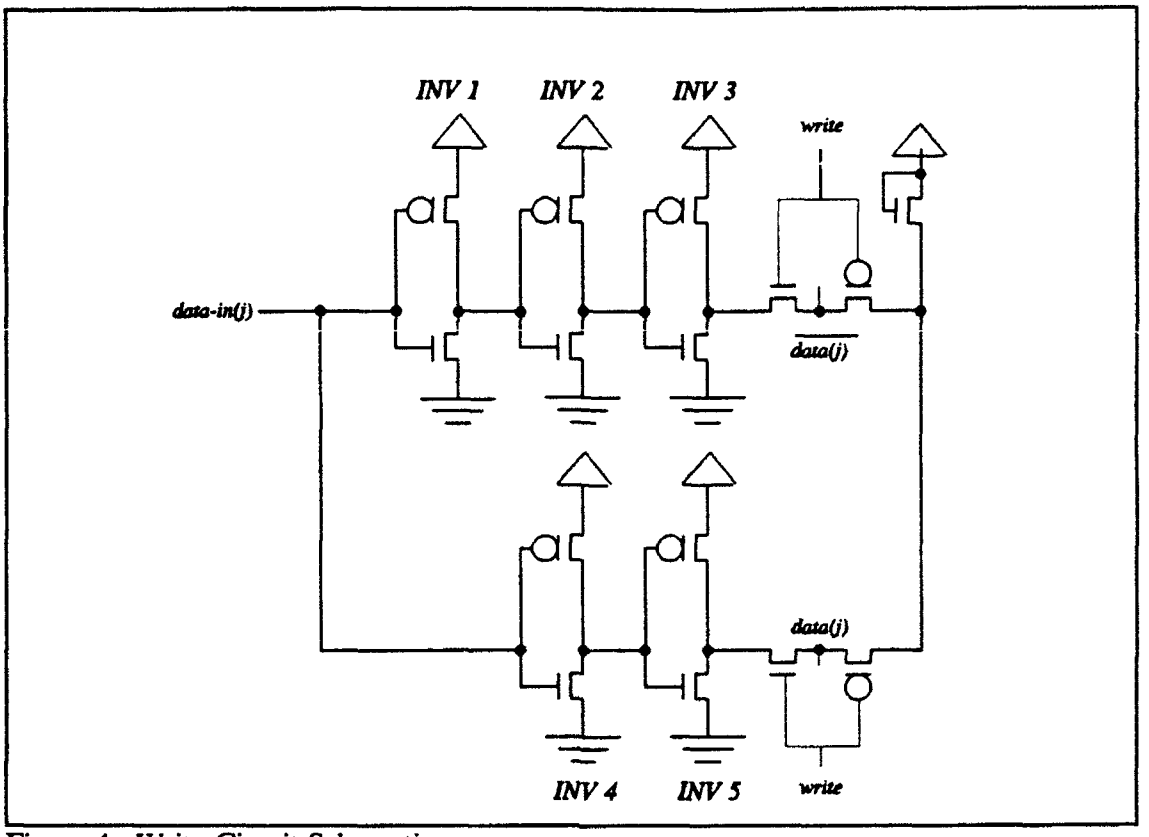

Figure 4. Write-Circuit Schematic.

value of approximately 3.6 V. Since the *data(j)* and $\overline{data(j)}$ ports are precharged to approximately 3.6 V, the rise propagation delay is relatively short and measured to be less than 0.6 ns, while the fall propagation delay may require more time. The fall propagation delay initiates the write process into the SRAM array. For this reason, it is more closely measured than the rise propagation delay. The data-precharge delay is also referenced from the *write* port transition, and is the propagation delay necessary to drive either the *data(j)* or $\overline{data(j)}$ port to V_{dd}-V_{tn}.

Table 5 gives the data, $\overline{data}$, and data-precharge delays. The data and $\overline{data}$ delays are functions of the number of words in the SRAM array, but not the number of SRAM cells in a word. The data and $\overline{data}$ fall and data-precharge delays linearly increase as the number of words in the SRAM array, and the corresponding load, linearly increases.

Table 3

SRAM Layout Dimensions.

	Width (λ)	Height (λ)
Decode Circuit	211	50
Driver Circuit	32	150
Read Circuit	32	50
SRAM Cell	32	50
Write Circuit	32	200

To show this linear dependency, Table 6 provides the least-square analysis of the delays used to calculate the coefficient of determination, the offset, and the slope. The least-square analysis shows only a slight difference in offset and slope between data and $\overline{data}$. This result is expected. The data-write circuit uses inverters with the same transistor-gate dimensions to drive the same size load on the *data(j)* and $\overline{data}(j)$ ports. Table 6 also provides the least-square analysis for the critical-path delays to be discussed in the remainder of Section 3.3.

3.3.2 Address-Driver Description. The address driver accepts the *address-in* port stimuli from the host to form the *address* and $\overline{address}$ stimulus usable by the address-decode circuit. It consists of *K* drivers. Figure 5 shows a driver schematic for *address-in(k)*, and Table 7 gives the transistor-gate dimensions. The value placed on *address-in(k)* is inverted twice to form the value on *address(k)* and three times to form the value on $\overline{address}(k)$. Except for INV1, the inverters are cascaded so that each progressive inverter has n-type and p-type transistor widths twice as large as the previous inverter. Cascading is necessary to drive the 8 fF/word capacitance on the *address(k)* and $\overline{address}(k)$ ports.

Table 4

Write Circuit Transistor-Gate Dimensions for the SRAM Architecture.

Write Circuit Gate Length = 2λ	Gate Width (λ)
INV1 P	4
INV1 N	3
INV2 P	9
INV2 N	6
INV3 P	18
INV3 N	12
INV4 P	9
INV4 N	6
INV5 P	18
INV5 N	12
Data Line P	3
Data Line N	7
$\overline{\text{Data}}$ Line P	3
$\overline{\text{Data}}$ Line N	7
Vdd P	3

The time to form the *address(k)* and $\overline{address(k)}$ ports also has the potential to be a critical-path delay for both the read and write instructions. The address and $\overline{\text{address}}$ delays are referenced from the *address-in(k)* port. The address rise delay is the propagation delay necessary for the *address(k)* port to reach the 50% point in its transition from ground to V_{dd}. The address fall delay is the propagation delay necessary for the *address(k)* port to reach the 50% point in its transition from V_{dd} to ground. The $\overline{\text{address}}$ rise delay is the propagation delay necessary for the $\overline{address(k)}$ port to reach the 50% point in its transition from ground to V_{dd}, and the $\overline{\text{address}}$ fall delay is the propagation delay necessary

Table 5

Data, $\overline{Data}$, and Data Precharge Delays for the SRAM Architecture.

Write Circuit	data delay (ns) reference: *write* 50% pt range: 0 to 3.7 V	$\overline{data}$ delay (ns) reference: *write* 50% pt range: 0 to 3.7 V	precharge delay (ns) reference: *write* 50% pt range: 0 to 3.6 V (unless otherwise stated)
4 words	$t_r < 0.5$ $t_f = 0.1$	$t_r < 0.5$ $t_f = 0.1$	$t_r = 6.5$
8 words	$t_r < 0.5$ $t_f = 0.3$	$t_r < 0.5$ $t_f = 0.3$	$t_r = 9.9$
16 words	$t_r < 0.5$ $t_f = 0.5$	$t_r < 0.5$ $t_f = 0.4$	$t_r = 16.3$
32 words	$t_r < 0.6$ $t_f = 0.6$	$t_r < 0.6$ $t_f = 1.0$	0 - 3.4 V $t_r = 28.6$
64 words	$t_r < 0.5$ $t_f = 1.4$	$t_r < 0.6$ $t_f = 1.8$	0 - 3.3 V $t = 50.6$
128 words	$t_r < 0.6$ $t_f = 2.5$	$t_r < 0.6$ $t_f = 3.4$	0 - 2.6 V $t_r = 99.0$

for the $\overline{address(k)}$ port to reach the 50% point in its transition from V_{dd} to ground.

Table 8 gives the address and $\overline{address}$ delays for simulated SRAM array lengths. The address and $\overline{address}$ delays are functions of the number of words in SRAM array, but not the number of SRAM cells in a word. Though the drivers are nearly equal in size for the *address* and $\overline{address}$ ports, the n-channel transistor size allowed a larger current carrying capability than the p-channel transistor of equal size. This explains the shorter fall delays as compared to the rise delays. The fall delay disables the words not selected by *address-in*, so to reduce the chance of multiple enabled words, the fall delays are designed to take less time than the rise delays.

Table 6 shows that the rise and fall delays linearly increase as the number of words in the SRAM array linearly increases. Since the *address(k)* and $\overline{address(k)}$ ports are

Table 6

Least-Square Analysis for the SRAM Architecture.

Delay	Coefficient of Determination	Offset (ns)	Slope (ns/word)
Data Fall Delay	0.99	0.11	0.02
$\overline{Data}$ Fall Delay	0.96	0.06	0.03
Data Precharge Delay	0.99	4.05	0.74
Address Rise Delay	0.99	0.25	0.04
$\overline{Address}$ Rise Delay	0.99	1.07	0.04
Address Fall Delay	0.99	0.61	0.01
$\overline{Address}$ Fall Delay	1.00	0.64	0.02
Address Decode Delay	0.89	2.73	0.06
Read Rise Delay	0.99	2.56	0.08
Read Fall Delay	0.99	4.90	0.59
Write Time for an 8-bit Word	0.99	4.27	0.04
Write Time for a 16-bit Word	0.99	4.57	0.04
Write Time for a 32-bit Word	0.99	5.27	0.04
Read Time for an 8-bit Word	0.99	3.63	0.11
Read Time for a 16-bit Word	0.99	4.03	0.11
Read Time for a 32-bit Word	0.99	4.63	0.11

equally loaded and are driven by equally sized inverters, their slopes are equal. The offset difference is attributed to the difference in the number of inverters found within the driver required to form *address(k)* and $\overline{address}(k)$ from *address-in(k)*.

3.3.3 Address-Decode Circuit Description. The address-decode circuit accepts the *address* and $\overline{address}$ port stimulus from the address driver to form an *I* element *word-select* port used to select a SRAM array word. The address-decode circuit consists of *I* decode circuits [13]. Figure 6 shows the decode-circuit schematic corresponding to the thirty-second

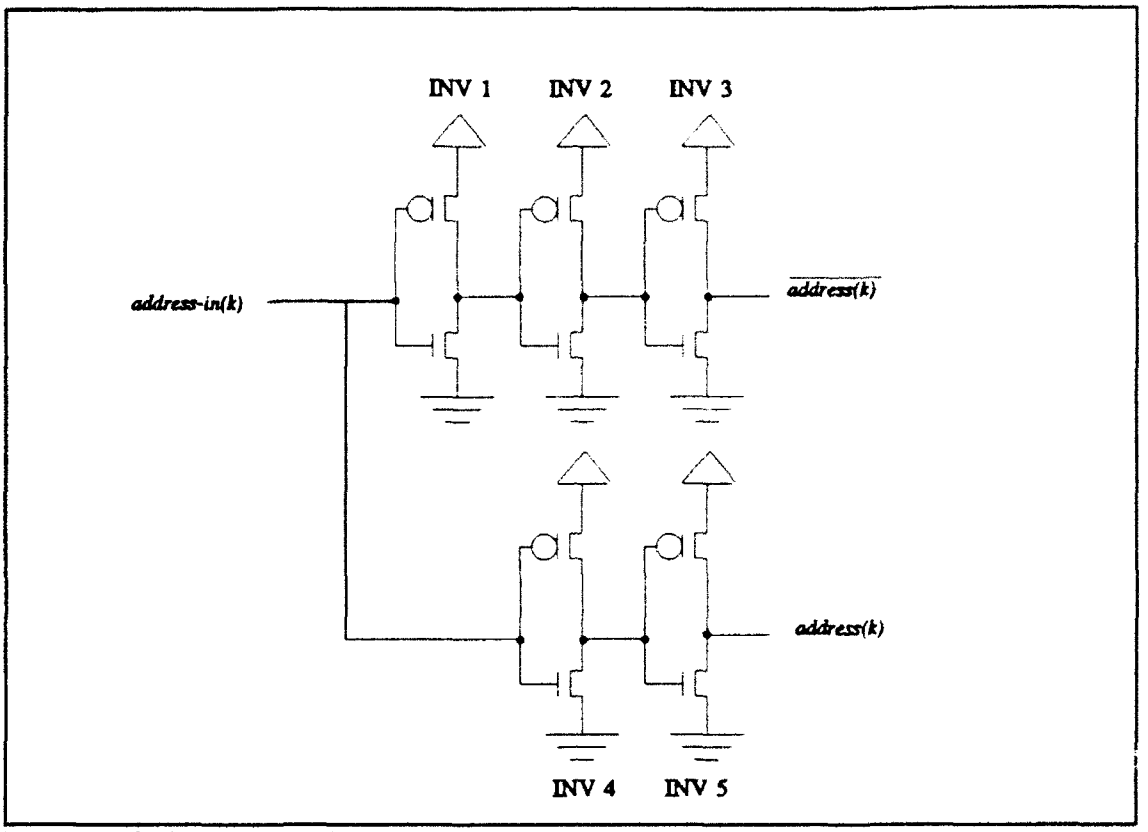

Figure 5. Driver Schematic.

word of a thirty-two word SRAM, and Table 9 gives the transistor-gate dimensions. The decode-circuit width is constrained by the *address* and $\overline{address}$ ports second-level metal runs. The decode-circuit height is constrained to the height of the SRAM cell, which is a maximum of 50 λ. The decode circuit could be designed shorter, but the design would force long metal runs to connect the *word-select* port to the SRAM array. The three inverters are cascaded so that each progressive inverter has n-type and p-type transistor widths two times as large as the previous inverter. The cascading is necessary to drive the 7 fF/word capacitance found on the *word-select(i)* port. Ideally, a p-type to n-type transistor gate ratio of 2 would yield a rise delay comparable to the fall delay, but the circuit layout dimension constraints imposed by the SRAM cell layout dimensions made the ideal ratio impractical to lay out.

Table 7

Driver Transistor-Gate Dimensions.

Address Drive Circuit Gate Length = 2λ	Gate Width (λ)
INV1 P	6
INV1 N	3
INV2 P	7
INV2 N	6
INV3 P	14
INV3 N	12
INV4 P	7
INV4 N	6
INV5 P	14
INV5 N	12

The circuit consists of a pseudo-NMOS NAND-gate driving three inverters. The NAND-gate consists of seven transistors. The single p-channel transistor gate is grounded causing the device to continuously conduct. The six n-channel transistors perform the NAND logic operation. Unless the NAND-gate is grounded through the six n-channel transistors, the p-channel transistor switches the three inverters to disable *word-select(31)*. When *address-in* is 11111 and *word-enable* is disabled, the five transistors connected to the *address* port precharge, thus placing ground at the source of the transistor with the *word-enable* gate input. When *word-enable* is enabled, the NAND-gate output is grounded thus switching the three inverter and enabling *word-select(31)*.

Table 8 gives the decode delay. The decode delay is referenced from the *address-in(k)* port transition from ground to V_{dd}, and is the propagation delay necessary to ground the decode circuit NAND-gate n-channel transistors except the word-enable transistor.

Table 8

Address, $\overline{\text{Address}}$, and Address Decode Delays
for the SRAM Architecture.

Address Delays	address delay (ns) reference: *address-in(k)* 50% pt range: 0 to 5 V	$\overline{\text{address}}$ delay (ns) reference: *address-in(k)* 50% pt range: 0 to 5 V	decode delay (ns) reference: *address-in(k)* 50% pt range: 2 to 0.1 V
4 words	t_r= 0.5 t_f= 0.6	t_r= 1.3 t_f= 0.9	t_f = 2.8
8 words	t_r= 0.6 t_f= 0.7	t_r= 1.4 t_f= 1.0	t_f = 3.1
16 words	t_r= 0.9 t_f= 0.8	t_r= 1.7 t_f= 1.2	t_f = 3.6
32 words	t_r= 1.5 t_f= 1.0	t_r= 2.2 t_f= 1.5	t_f = 4.8
64 words	t_r= 3.0 t_f= 1.3	t_r= 3.4 t_f= 1.9	t_f = 6.8
128 words	t_r= 5.7 t_f= 2.0	t_r= 6.0 t_f= 3.0	t_f =10.0

This data was measured to ensure the proper operation of the circuit, and the selection of 0.1 V to represent a grounded signal was arbitrary.

Table 10 gives the word-select delays. The word-select rise and fall delays are referenced from the *word-enable* port. The word-select rise delay is the propagation delay for the *word-select(i)* port to reach the 50% point in its transition from ground to V_{dd}. The word-select fall delay is the propagation delay for the *word-select(i)* port to reach the 50% point in its transition from V_{dd} to ground. The decode circuit word-select delays are functions of the number of SRAM cells in a word, but are not functions of the number of words in the SRAM array. This requires additional explanation. The research assumes that the ports driven by the host are of sufficient current to drive the SRAM array. Therefore, the *word-enable* port is defined to transition at a fixed time no matter the number of words in the SRAM array. The word-select delays linearly increase as the number of SRAM cells in a word linearly increases. Using the least-square analysis, the word-select rise-delay offset

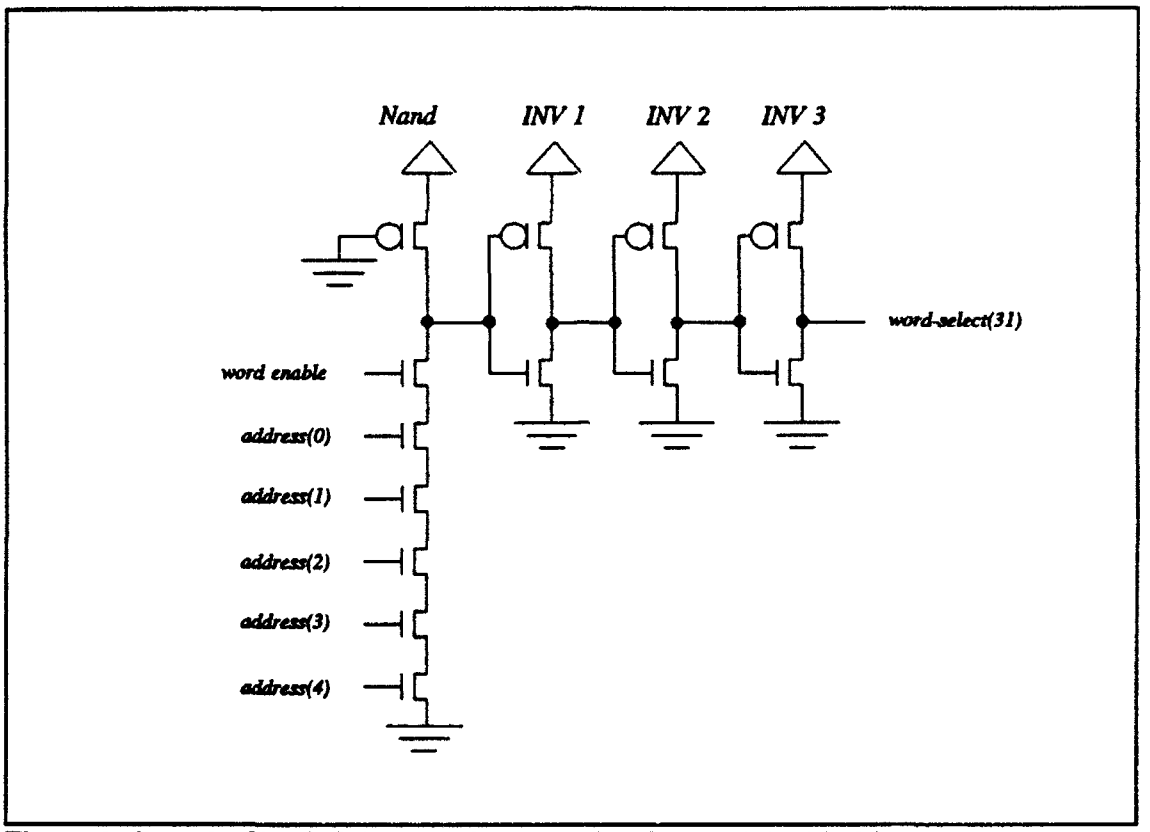

Figure 6. Decode-Circuit Schematic for the Thirty-Second Word in Memory.

is 1.35 ns; the slope is 0.04 ns/SRAM cell, and the coefficient of determination is 0.99. The word-select fall-delay offset is 1.65 ns; the slope is 0.02 ns/SRAM cell, and the coefficient of determination is 0.99.

3.3.4 SRAM-Array Description. Figure 7 shows the SRAM array. The SRAM array contains *I* times *J* SRAM cells, identified as *SRAM cell(0, 0)* to *SRAM cell(I-1, J-1)*. Figure 8 shows the SRAM-cell schematic. The SRAM-cell layout dimensions affect the dimensions of the remaining circuits. The SRAM-cell width constrains the width of the read and write circuit, and the SRAM-cell height constrains the height of the decode circuit. The SRAM cell consists of six transistors with transistor-gate widths all equal to 3 λ.

To write a 0 into *SRAM cell(i, j)*, *word-select(i)* is enabled. After the maximum value of the data and $\overline{\text{data}}$ delays, the data-write circuit drives *data(j)* to ground and $\overline{data}(j)$ to

Table 9

Decode Circuit Transistor-Gate Dimensions.

Decode Circuit Gate Length = 2λ	Gate Width (λ)
Pseudo NMOS Nand-gate P	3
Pseudo NMOS Nand-gate N	16
INV1 P	4
INV1 N	3
INV2 P	8
INV2 N	6
INV3 P	16
INV3 N	12

Table 10

Word-Select Delays for the SRAM Architecture.

Decode Circuit	word-select delay (ns) 8 bits reference: *word-enable* 50% pt range: 0 to 5 V	word-select delay (ns) 16 bits reference: *word-enable* 50% pt range: 0 to 5 V	word-select delay (ns) 32 bits reference: *word-enable* 50% pt range: 0 to 5 V
For All Word Lengths	t_r= 1.7 t_f= 1.8	t_r= 2.0 t_f= 1.9	t_r= 2.7 t_f= 2.2

V_{dd}-V_{tn}. At this time, *data(j)* grounds node *A* and $\overline{data(j)}$ sets node *B* to V_{dd}-V_{tn}. Nodes *A* and *B* are inputs to inverters and the grounded node *A* is inverted to V_{dd} at node *B* pulling up node *B* from V_{dd}-V_{tn}. V_{dd} at node *B* is inverted setting node *A* to ground. This establishes a steady-state condition which remains after *word-select(i)* is disabled. To store a 1 into

SRAM cell(i, j), *data(j)* is set to V_{dd}-V_{tn} and $\overline{data(j)}$ is grounded. The storage process is similar to the discussion above. The time to change the content of *SRAM cell(i, j)* once the *word-select(i)* port has reached the 50% point in its transition from ground to V_{dd} is approximately 1.5 ns.

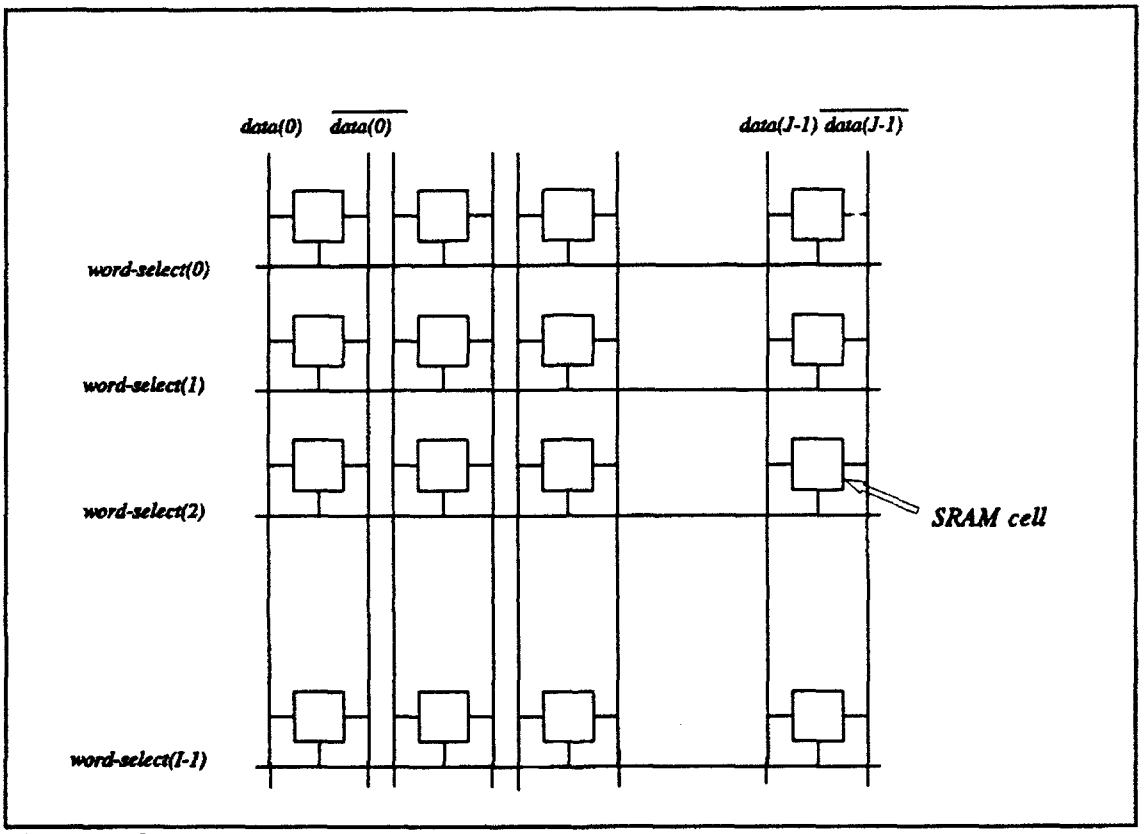

Figure 7. SRAM Array.

To read the content of *SRAM cell(i,j)*, both *data(j)* and $\overline{data(j)}$ are precharged to V_{dd}-V_{tn}. After the data-precharge delay, *word-select(i)* is enabled. If the content of *SRAM cell(i, j)* is a 0, then *data(j)* is grounded and $\overline{data(j)}$ remains at V_{dd}-V_{tn}. If the content of *SRAM cell(i, j)* is a 1, then *data(j)* remains at V_{dd}-V_{tn}, while $\overline{data(j)}$ is grounded.

3.3.5 Data-Read Circuit Description. The final component of the SRAM is the data-read circuit. The SRAM array and the data-write circuit drive the data-read circuit using the *data* and $\overline{data}$ ports. Like the data-write circuit, the data-read circuit is made of *J* read

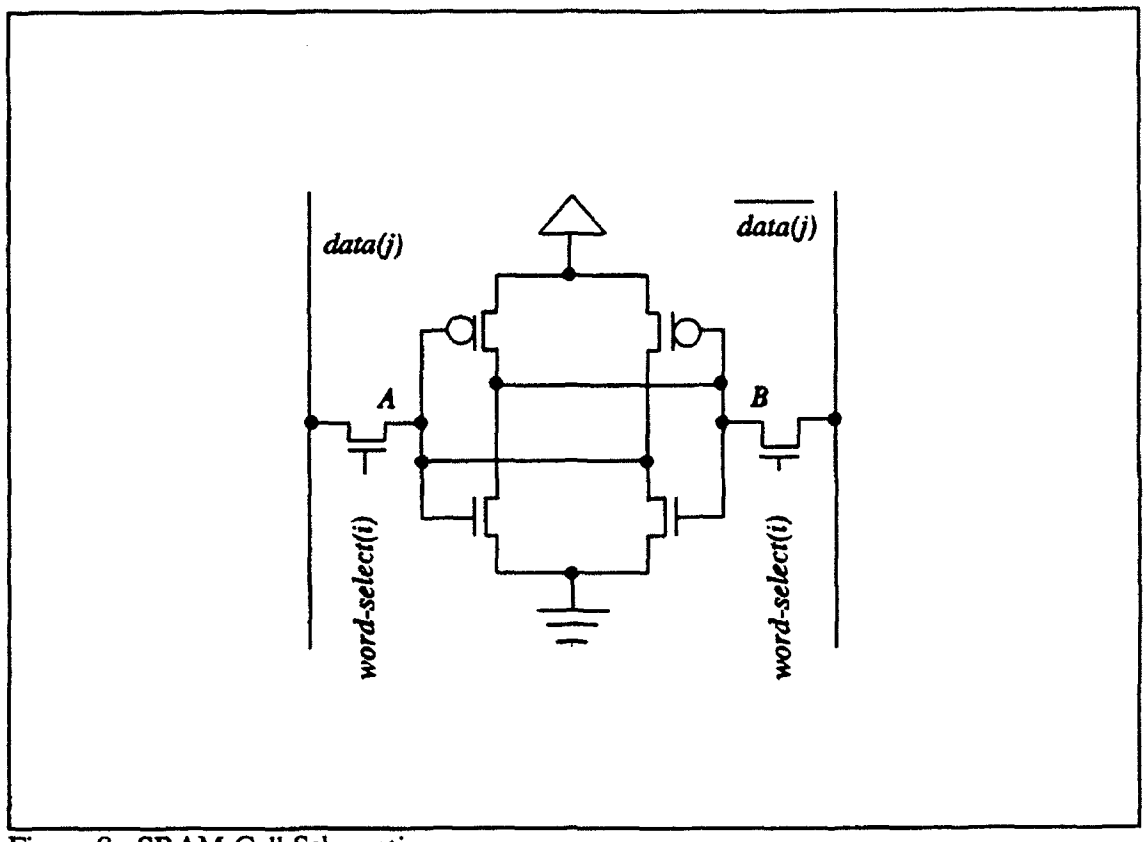

Figure 8. SRAM-Cell Schematic.

circuits.

Figure 9 shows the read-circuit schematic, and Table 11 gives the transistor-gate dimensions [13, 18]. The read circuit is a differential-sensing amplifier that detects small changes between *data(j)* and $\overline{data(j)}$ to identify the appropriate value for *data-out(j)*. Under normal operating conditions, the read instruction begins by precharging both *data(j)* and $\overline{data(j)}$ to V_{dd}-V_{tn}. This, in turn, partially turns on transistors *Q3* and *Q4* and pulls *data-out(j)* to a low value. As *data-out(j)* goes low, transistors *Q1* and *Q2* begin to turn on which pulls *data-out(j)* to a high value. A steady-state condition between ground and V_{dd} eventually is established.

When *word-select(i)* is enabled, if *SRAM cell(i, j)* contains a 0, *data(j)* begins to change from V_{dd}-V_{tn} to ground, while $\overline{data(j)}$ remains relatively unchanged. The current

through *Q3* decreases which lowers the current through *Q1* and *Q2* and increases the current through *Q4*. Lowering the current through *Q2* and increasing the current through *Q4* causes the read circuit to sink more current. If *SRAM cell(i, j)* contains a 1, *data(j)* remains relatively unchanged while $\overline{data(j)}$ begins to change from V_{dd}-V_{tn} to ground. The current through *Q4* decreases which increases the current through *Q3*, *Q1*, and *Q2*. Increasing the current through *Q2* and lowering the current through *Q4* causes the circuit to source more current. Proper transistor ratios will create a *data-out(j)* value that can be used to represent a 0, though the value is not grounded.

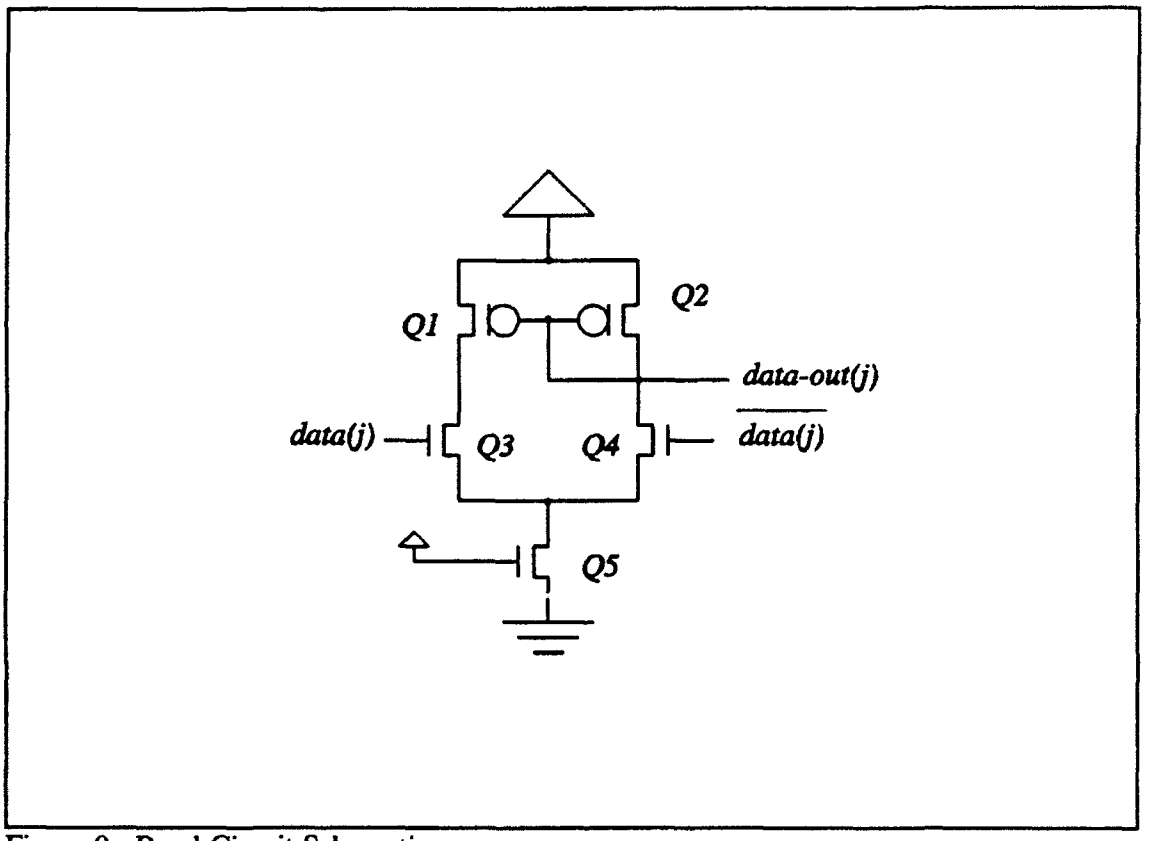

Figure 9. Read-Circuit Schematic.

Table 12 gives the read delays. The read rise and fall delays are referenced from the *word-enable* port. The read rise delay is the propagation delay for *data-out(j)* to reach the 50% point in its transition from a steady-state voltage of approximately 1.5 V to a high value

of approximately 4.2 V. The read fall delay is propagation delay for *data-out(j)* to reach the 50% point in its transition from a high value of approximately 4.2 V to steady-state. The transition from the steady-state to a low value using external stimulus was recorded to be consistently less than 0.5 ns. Again the read delays linearly increase as the number of words linearly increases.

Table 11

Read Circuit Transistor-Gate Dimensions

Read Circuit Gate Length = 2λ	Gate Width (λ)
Q1	7
Q2	7
Q3	4
Q4	4
Q5	5

The read delays are functions of the number of SRAM cells in a word and the number of words in the SRAM array. The SRAM was simulated for an 8 SRAM-cell word. The 16 SRAM-cell word read delays were calculated by adding the difference between the 8-bit and 16-bit word-select rise delays, presented in Table 10, to the 8 SRAM-cell read delays. The 32 SRAM-cell word read delays were calculated by adding the difference between the 8-bit and 32-bit word-select rise delays to the 8 SRAM-cell read delays. To show the linear dependency, Table 6 provides the least-square analysis. The read rise-delay slope is smaller than the fall-delay slope. The rise delay is externally driven, while the fall delay is a function of the *data(j)* and $\overline{data(j)}$ precharge delay.

Table 12

Read Delays for the SRAM Architecture.

Read Circuit	read delay (ns) reference: *word-enable* 50% pt range: 1.3 to 4.1 V
4 words	t_r= 2.8 t_f= 6.7
8 words	t_r= 3.2 t_f= 9.2
16 words	t_r= 3.8 t_f=14.3
32 words	t_r= 5.1 t_f=24.5
64 words	t_r=7.2 t_f=43.3
128 words	t_r=12.3 t_f=79.8

3.4 Read and Write-Instruction Access-Time Characteristic Equations.

The SRAM can execute two operations, write and read. Figure 10 shows the timing diagram to execute both. When the host writes data into the SRAM, it places the address of the desired word location for storage onto the *address-in* port, the data to be stored in the SRAM array onto the *data-in* port, and enables the *write* port. After the maximum delay necessary to decode the address, the *word-enable* port is enabled. After a delay necessary to write the word into the SRAM array, the host can disable the *word-enable* and *write* ports, then remove the address and data stimulus.

The time required to write a word into the SRAM array is the sum of the maximum value of the address and $\overline{\text{address}}$ delays found in Table 8, the word-select rise delay found in Table 10, and the time to store the data into SRAM cell. Table 13 gives this data. The write-access time linearly increases as the number of words in the SRAM array and the number of SRAM cells in a word increase. This linear increase is expected and represents a linear increase in load.

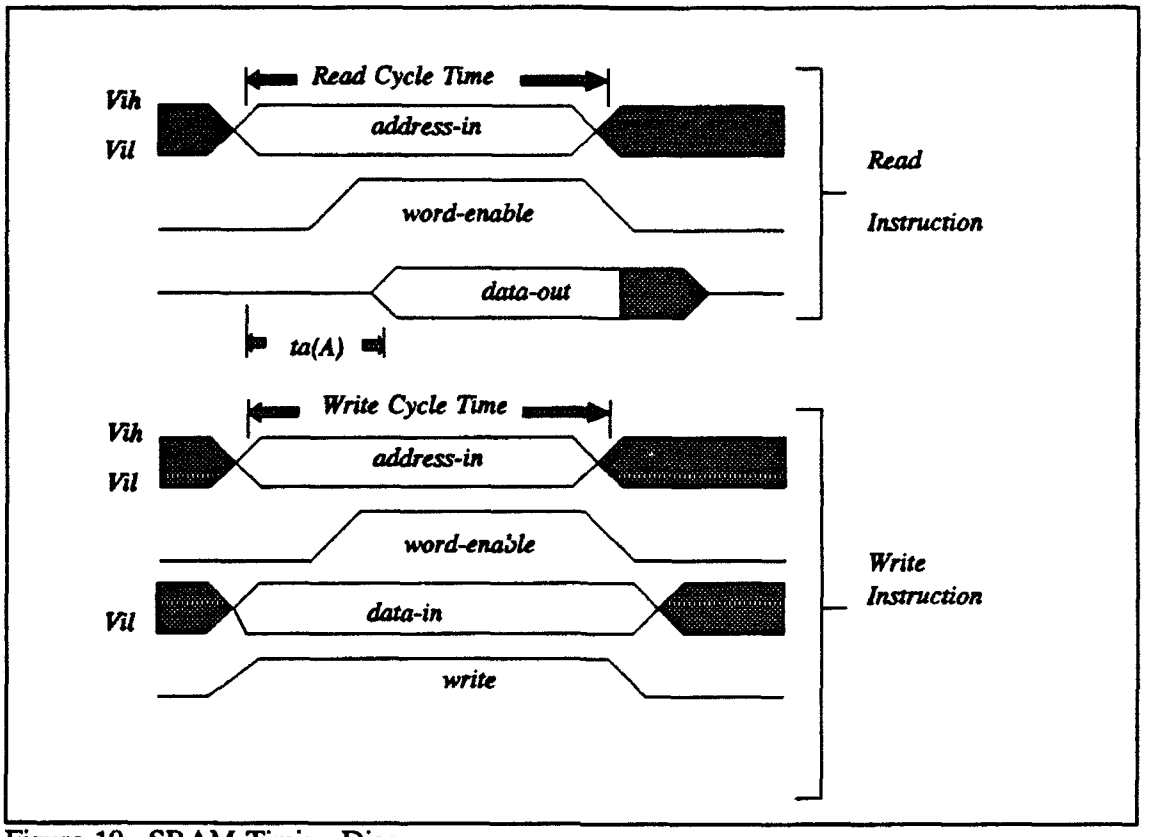

Figure 10. SRAM Timing Diagram.

To show this linear dependency, Table 6 provides the least-square analysis of the delays. The analysis is performed for 8, 16, and 32 SRAM-cell word lengths. From a least-square analysis of the three offsets, a single equation characterizing the write instruction access time is formed. Equation (46) approximates the SRAM write-access time.

$$\tau_{write}=(3.92+0.042T(s)+0.038I)\ ns \tag{46}$$

As expected, the write-access time depends upon the number of SRAM cells in a word and the number of words in the SRAM array.

The percentage difference between the equation and the tabulated write-access times is measured to be less than 2.0% for all simulated SRAM array sizes. The percentage difference was calculated by dividing the difference between the measured and calculated delay by the measured delay.

Table 13

SRAM Write Access Time.

Write	8 bits (ns)	12 bits (ns)	32 bits (ns)
4 words	1.3+1.7+1.5 = 4.5	1.3+2.0+1.5 = 4.8	1.3+2.7+1.5 = 5.5
8 words	1.4+1.7+1.5 = 4.6	1.4+2.0+1.5 = 4.9	1.4+2.7+1.5 = 5.6
16 words	1.7+1.7+1.5 = 4.9	1.7+2.0+1.5 = 5.2	1.7+2.7+1.5 = 5.9
32 words	2.2+1.7+1.5 = 5.4	2.2+2.0+1.5 = 5.7	2.2+2.7+1.5 = 7.4
64 words	3.4+1.7+1.5 = 6.6	3.4+2.0+1.5 = 6.9	3.4+2.7+1.5 = 7.6
128 words	6.0+1.7+1.5 = 9.2	6.0+2.0+1.5 = 9.5	6.0+2.7+1.5 =10.2

When the host reads data from the SRAM, it places the address of the desired word on the SRAM *address-in* port. After the maximum delay necessary to decode the address, the *word-enable* port is enabled. After the delay necessary to read the word and form the *data-out* port, the host disables the *word-enable* port, then removes the *address-in* port stimuli.

The time required to read a word from the SRAM array is the sum of the maximum value of the address and $\overline{address}$ delays, and the read delay found in Table 11. Table 14 gives this data. Not included is the delay associated with precharging the *data* and $\overline{data}$ ports. The precharge delay is assumed to have occurred and is excluded not because it is unimportant, but rather because the circuitry associated with it would normally be custom-designed for each SRAM array size. This custom-design approach is beyond the scope of this research.

The delay linearly increases as the number of words in the SRAM array and the number of SRAM cells in a word increase. Table 6 shows that the slopes are equal though the offset shifts to higher values as the number of SRAM cells in a word increases. From a

least-square analysis of the three offsets, a single equation characterizing the read-access time is formed. Equation (47) approximates the SRAM read-access time.

$$\tau_{read}=(3.328+0.041T(s)+0.11l)\ ns \tag{47}$$

The percentage difference between the equation and the tabulated read-access times is calculated to be less than 3.3% for all simulated SRAM-array sizes.

Table 14

SRAM Read Access Time.

READ	8 bits (ns)	16 bits (ns)	32 bits (ns)
4 words	1.3+2.8= 4.1	1.3+2.8+0.4 = 4.5	1.3+2.8+1.0 = 5.1
8 words	1.4+3.2= 4.6	1.4+3.2+0.4 = 5.0	1.4+3.2+1.0 = 5.6
16 words	1.7+3.8= 5.5	1.7+3.8+0.4 = 5.9	1.7+3.8+1.0 = 6.5
32 words	2.2+5.1= 7.3	2.2+5.1+0.4 = 7.7	2.2+5.1+1.0 = 8.3
64 words	3.4+7.2= 10.6	3.4+7.2+0.4= 11.0	3.4+7.2+1.0= 11.6
128 words	6.0+12.3= 18.3	6.0+12.3+0.4= 18.7	6.0+12.3+1.0 = 19.3

3.5 Summary

This chapter presented the static random-access memory which was designed to serve as a benchmark by which to compare the associative-memory architectures. The benchmark is used to compare read and write-access times as well as circuit layout dimensions. Both the read and write-access time equations are used in Chapter VII as a benchmark to compare against the associative-memory architectures. The component layout dimensions of Table 3 will be used in Chapter VII to calculate the size per SRAM cell of the SRAM.

IV. Word-Organized Fully-Parallel Associative Memory

4.1 Introduction

This chapter presents a word-organized fully-parallel associative memory, also called a content-addressable memory (CAM). The CAM was designed to demonstrate the relationship between BPI, RCI operations and an associative-memory organization that exhibits an ability to simultaneously compare every bit of the memory array to a comparand.

The chapter develops the characteristic equations to approximate the read, write, and BPI, RCI instruction execution times. The chapter begins by describing the CAM organization. It continues by describing the CAM architecture to the transistor level. In this description, potential critical paths are identified, and the tabulated delays are presented. The chapter continues by explaining the instruction execution procedures and concludes by providing the instruction execution-time equations.

Throughout the chapter, enhancements to the CAM architecture are mentioned. These enhancements allow the architecture to execute BPD operations and are necessary to compare different associative-memory organizations in their execution of a mix of associative-search operations.

For a complete VHDL description of the CAM, refer to Appendix C.

4.2 CAM Organization and General Operation Description

The CAM organization contains a comparand register, a mask register, a memory array called the CAM array, a word/search-select register, a search-results register, and an output register. Central to the CAM organization is the CAM array. The CAM array is a two-dimensional array of CAM cells with each cell containing the logic to perform a scalar

equivalence operation with one element from the comparand and the other from the CAM cell itself.

The CAM executes write, read, and associative-search instructions in either the word or phrase mode. Data can be written into the CAM array either one word at a time, to a subset of words, or to the entire array. Data can be retrieved from the CAM array one word at a time. The location of data storage and retrieval is defined by the CAM architecture and not by the host. The CAM can execute BPI, RCI operations as instructions. BPD operations are executed as a sequence of BPI, RCI instructions.

To execute a BPI, RCI instruction, the results of comparing a bit of the comparand to the corresponding bit-slice of the CAM array are logically ANDed with other bit-slice comparisons to form a final result, which is stored in the search-results register. Since the CAM can execute only BPI, RCI instructions, the data format does not affect the instruction procedure. Therefore, the comparand bits can be compared to the corresponding bit-slice in any order, including simultaneously.

The word mode divides a word into at least two fields. The first field is the valid-data field, which consists of a single bit, $T_{vdf}=1$. A stored 1 means the corresponding word is valid, while a stored 0 means the corresponding word is invalid. The remaining portion of a word can store database-record fields.

In the phrase mode, a word consists of at least three fields. The first field is the valid-data field. The second field is the phrase field. Since the number of words in a phrase equals eight for this research endeavor, this implies that T_{pf} equals 3. A stored value of 000 corresponds to the first word of a phrase. A stored value of 001 corresponds to the second word of a phrase, etc. The remaining CAM cells in the word can be partitioned into fields containing data.

4.3 CAM-Architecture Description

Figure 11 shows the CAM architecture. It consists of four components: the data-write circuit, the CAM array, the data-read circuit, and the word-select circuit. The comparand and mask registers of the CAM organization (not shown) drive the data-write circuit which in turn drives the CAM array. The word-select and search-results registers of the CAM organization are found within the word-select circuit. The data-read circuit drives the output register (not shown). These four components are generated from repetitions of less complicated circuits. This section presents, to the transistor level, these four components.

The CAM architecture interfaces with the host using five ports: *data-in*, *mask-in*, *write*, *data-out*, and *control-in*. The host supplies data to the CAM architecture using the N-element *data-in* port and retrieves data using the N-element *data-out* port, where N is the total number of CAM cells in a word. It uses the N-element *mask-in* port to identify the bit-slices of the CAM array involved in the instruction and uses the eight-element *control-in* port to select the instruction to execute. The host does not keep track of where data are stored within the CAM array, so no address is needed. The word-select circuit performs this function.

Within this chapter, each circuit description identifies potential critical-path delays for simulated word lengths of N equal to 12, 20, and 36 CAM cells and I equal to 8, 16, 32, 64, and 128 words. The 12, 20, and 36-CAM cell word lengths were chosen for simulation to represent an 8, 16, and 32-CAM cell data field appended with a single CAM-cell valid-data field and a three CAM-cell phrase field. Like the SRAM, insufficient computer memory prohibits the simulation of the entire CAM. The CAM architecture simulations require two circuit models. For simulations with memory lengths less than thirty-two, the CAM architecture models are similar to the SRAM. The CAM array is modified to include a single bit-slice of I-1 CAM cells and a single N-CAM cell word. The word-select circuit is provided

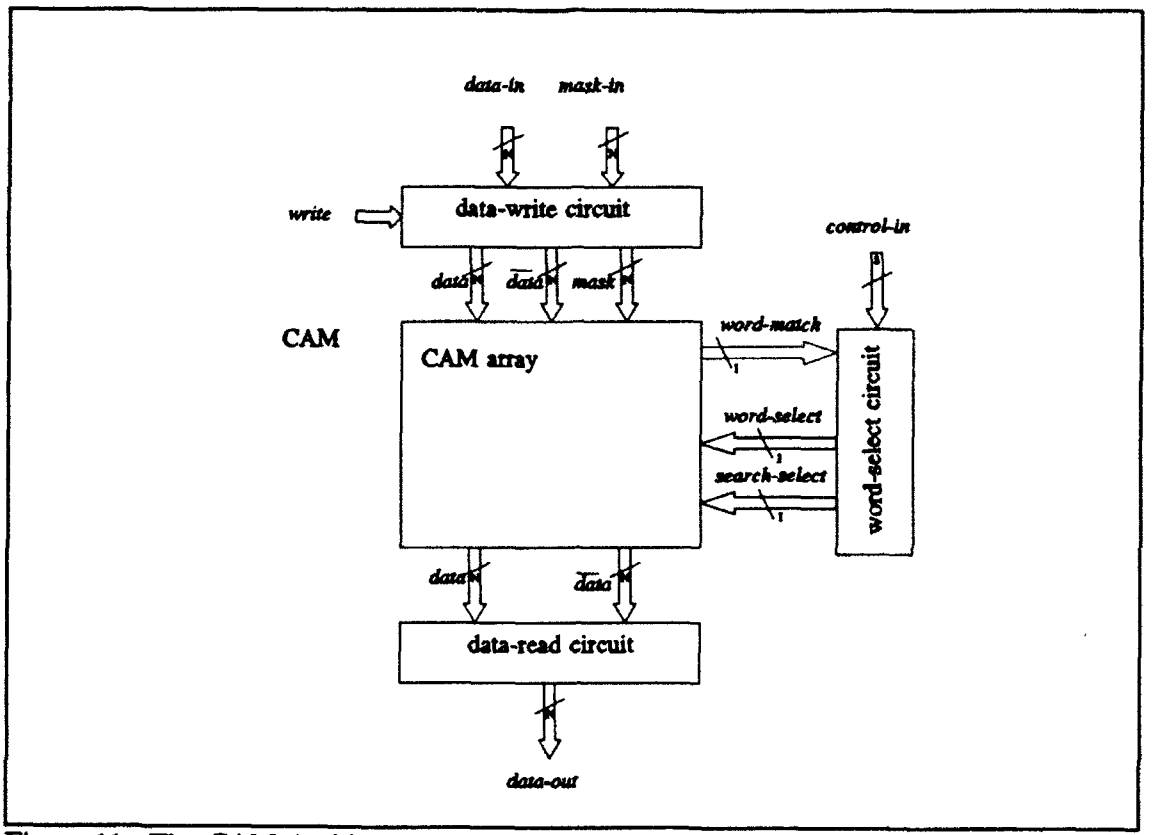

Figure 11. The CAM Architecture.

in its entirety. For memory lengths greater than thirty-two, the word-select circuit became too large for simulation, and a second circuit model was designed. The second circuit model extracts the critical-path delay circuitry found within the word-select circuit. This circuitry is loaded to properly simulate each instruction. When simulations mandate a static capacitance representation of a dynamically changing capacitive value, the worst case scenario is chosen so not to give optimistic performance results.

4.3.1 Data-Write Circuit Description. The data-write circuit accepts the *data-in*, *mask-in*, and *write* port stimuli from the host to form the *data*, $\overline{data}$, and *mask* port stimuli. The *data-in*, *data*, $\overline{data}$, *mask-in*, and *mask* ports consist of N elements, and an element within each port is identified as *data-in(n)*, *data(n)*, $\overline{data}(n)$, *mask-in(n)*, and *mask(n)*, respectively.

The data-write circuit consists of *N* write circuits. Figure 12 shows the write-circuit schematic for *data-in(n)*. Table 15 gives its layout dimensions, and Table 16 gives the transistor-gate dimensions. Table 15 also gives the layout dimensions for the other circuits discussed within this chapter. Contained within the CAM write circuit is the SRAM write circuit and additional mask circuitry. When *mask-in(n)* is enabled, the n^{th} bit-slice of the CAM array can store data; otherwise, the storage process is blocked.

The write circuit inverters are cascaded so that each progressive inverter has n-type and p-type transistor-gate widths three times as large as the previous inverter, except for the p-type transistors interfacing with *data(n)*, $\overline{data(n)}$, and *mask(n)*. These transistor-gate widths are approximately six times larger than the previous inverter p-type transistor-gate widths to compensate for the in-line n-type transistor. The CAM write circuit INV 3 and 5 transistor-gate widths are approximately six times larger than SRAM write circuit INV 3 and 5 transistor-gate widths. Though the capacitive values found on *data(n)* and $\overline{data(n)}$ are similar when writing to a single word, the CAM architecture is designed to write to multiple words, which mandates larger drivers.

The time to form *data(n)*, $\overline{data(n)}$, and *mask(n)* has the potential to be a critical-path delay. The data and $\overline{data}$ delays are defined in Chapter III. The mask delays are referenced from *mask-in* port. The mask rise delay is the propagation delay necessary for the *mask(n)* port to reach the 50% point in its transition from ground to V_{dd}. The mask fall delay is the propagation delay necessary for the *mask(n)* port to reach the 50% point in its transition from V_{dd} to ground.

Table 17 gives the data, $\overline{data}$, mask, and data-precharge delays. These delays are functions of the number of words in the CAM array, and except for the precharge delay, they linearly increase as the number of words in CAM array linearly increases. The data-precharge voltage range degrades for CAM-array lengths greater than about 64 words, so its delay

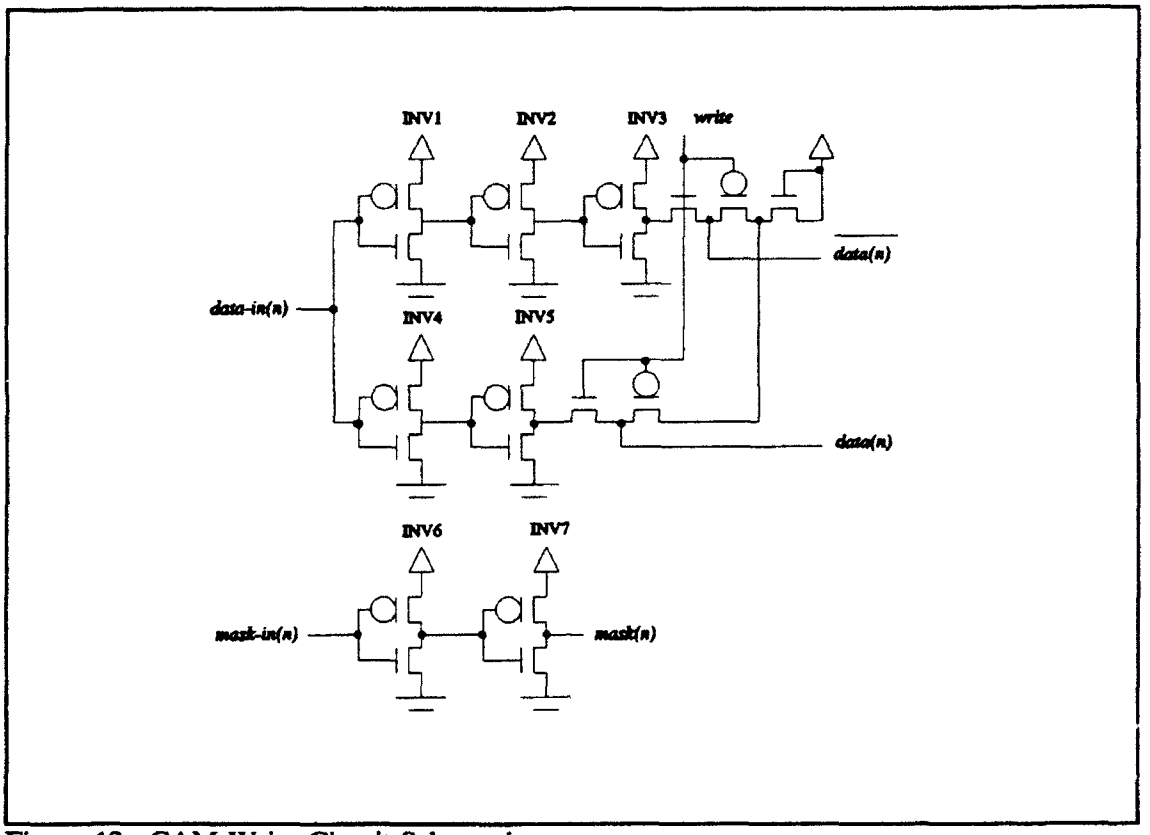

Figure 12. CAM Write-Circuit Schematic.

becomes increasingly nonlinear. As will be discussed later in the chapter, the degraded precharge voltage modifies the performance of the data-read circuit.

To verify that the precharge circuit causes the degraded voltage range, it was redesigned for a single memory-array size and re-simulated. The redesigned precharge circuit restored the voltage range between 0 and 3.2 V, and the data-precharge delay returned to its linear progression. The data acquired from the redesigned precharge circuit was not used. The purpose of the chapter is to obtain characteristic equations of the instruction-set execution times using repetitions of a single write circuit, word-select circuit, and read circuit and a variety of memory-array sizes. Modifying the write circuit for each memory array size deviates from this research approach.

Table 15

CAM Layout Dimensions.

	Width (λ)	Height (λ)
CAM Cell	36	83
Control-1 Driver	18	104
Control-2 Driver	32	150
Match Pre-Charge Circuit	33	83
MMR	66	332
Phrase Select Circuit	803	664
Read Circuit	36	50
Write Circuit	36	387

Table 18 shows that data and $\overline{data}$ fall delay offsets and slopes are approximately equal. The load on *data(n)* is 14 fF/word, while the load on $\overline{data}(n)$ is 15 fF/word, so the observation is reasonable. Table 18 also provides the least-square analysis for the other critical-path delays presented in this section.

4.3.2 CAM-Array Description. Figure 13 shows the CAM array. It contains *I* times *N* CAM cells, identified as *CAM cell(0, 0)* to *CAM cell (I-1, N-1)*. It contains *I*-element *word-select*, *search-select*, and *word-match* ports identified as *word-select(0)* to *word-select(I-1)*, *search-select(0)* to *search-select(I-1)*, and *word-match(0)* to *word-match(I-1)*. It contains *N*-element *data*, $\overline{data}$, and *mask* ports, using the same nomenclature as defined for the data-write circuit. The word-select circuit drives the *word-select* and *search-select* ports. Either the data-write circuit or the CAM array drives the *data* and $\overline{data}$ ports. The data-write circuit drives the *mask* port. In turn, the CAM-array *word-match* port drives the word-select circuit *word-match* port.

Table 16

Write Circuit Transistor-Gate Dimensions for the CAM Architecture.

Write Circuit Gate Length = 2λ	Gate Width (λ)
INV1 P	5
INV1 N	3
INV2 P	15
INV2 N	9
INV3 P	96
INV3 N	27
INV4 P	15
INV4 N	9
INV5 P	96
INV5 P	27
INV6 P	9
INV6 N	6
INV7 P	96
INV7 N	27
Data Line P	8
Data Line N	5
$\overline{\text{Data}}$ Line P	8
$\overline{\text{Data}}$ Line N	5
Vdd P	4

Figure 14 shows the CAM cell schematic, and Table 19 gives the CAM cell transistor-gate dimensions. The CAM architecture writes data into and reads data from the CAM cell in a similar manner to the SRAM. The only difference is the necessity to enable both *mask(n)* and *word-select(i)*.

Table 17

Data, $\overline{Data}$, Mask, and Data Precharge Delays
for the CAM Architecture.

Cam Write Circuit	data delay (ns) reference: *write* 50% pt	$\overline{data}$ delay (ns) reference: *write* 50% pt	mask delay (ns) reference: *mask_in* 50% pt range: 0 to 5.0 V	precharge delay (ns) reference: *write* 50% pt
8 words	0 - 3.9 V t_r<) t_f= 0.1	0 - 3.9 V t_r<2.0 t_f= 0.1	t_r= 0.7 t_f= 1.4	0 to 3.6 V t_r= 5.8
16 words	0 - 3.8 V t_r<2.0 t_f= 0.2	0 - 3.8 V t_r<2.0 t_f= 0.2	t_r= 0.9 t_f= 1.6	0 to 3.4 V t_r= 8.1
32 words	0 - 3.8 t_r<2.0 t_f= 0.3	0 - 3.8 t_r<2.0 t_f= 0.3	t_r= 1.0 t_f= 1.9	0 to 3.2 V t_r=15.7
64 words	0 - 3.8 V t_r<2.0 t_f= 0.4	0 - 3.8 V t_r<2.0 t_f= 0.5	t_r= 1.2 t_f= 2.2	0 to 3.0 V t_r=18.5
128 words	0 - 3.8 V t_r<2.0 t_f= 0.8	0 - 3.8 V t_r<2.0 t_f= 0.8	t_r= 2.0 t_f= 3.2	0 to 2.3 V t_r=26.0

To perform an equivalence search on the content of *CAM cell(i, n)*, *word-match(i)* is precharged to V_{dd}-V_{tn}, the data stimulus is placed onto *data(n)*, the inverse of the data stimulus is placed onto $\overline{data(n)}$, the mask stimulus is placed onto *mask(n)*, and *search-select(i)* is enabled. After the search is complete, *search-select(i)* is disabled, then the data and mask stimuli can be removed.

Suppose the host places a 0 onto *data-in(n)*, a 1 onto *mask-in(n)*, and enables the *write* port. Also suppose *CAM cell(i, n)* contains a 0, so a match should exist. Since *data-in(n)* is 0, *data(n)* is grounded, and $\overline{data(n)}$ is set to V_{dd}-V_{tn}. Since $\overline{Q}$ is set to V_{dd}, the grounded *data(n)* disconnects the search circuitry allowing *word-match(i)* to remain precharged. Now suppose the host places a 1 onto *data-in(n)*. *Data(n)* is set to V_{dd}-V_{tn}, and $\overline{data(n)}$ is grounded. Now *data(n)* causes *word-match(i)* to be grounded. The equivalence

Table 18

Least-Square Analysis for the CAM Architecture.

Delay	Coefficient of Determination	Offset (ns)	Slope (ns/word)
Data\$\overline{\text{Data}}$ Fall Delay	0.98\ 0.98	0.09\ 0.10	0.01\ 0.01
Mask Rise\Fall Delay	0.98\ 0.99	0.65\ 1.35	0.01\ 0.01
Read Rise Delay; 12\20\36-bit words	0.98\ 0.98\ 0.96	10.36\11.26\12.28	0.33\ 0.33\ 0.30
Read Fall Delay; 12\20\36-bit words	0.70\ 0.74\ 0.75	11.24\11.31\11.40	0.11\ 0.11\ 0.10
Control Rise\Fall Delay	1.00\ 0.99	0.91\ 0.83	0.09\ 0.04
$\overline{\text{Control}}$ Rise Delay	0.99	1.29	0.07
$\overline{\text{Control}}$ Fall Delay	0.99	1.30	0.03
Control(4) Rise/Fall Delay	0.97\0.84	-1.11\1.60	0.24\0.03
Word-Select Rise Delay; 12\20\36-bit Words	1.00\ 1.00\ 1.00	4.02\ 4.39\ 4.99	0.26\ 0.25\ 0.26
Word-Select Fall Delay; 12\20\36-bit words	0.92\ 0.93\ 0.91	2.812\ 3.10\ 3.53	0.02\ 0.02\ 0.02
Search-Select Rise Delay; 12\20\36-bit words	1.00\ 1.00\ 1.00	4.79\ 4.95\ 5.21	0.26\ 0.26\ 0.26
Search-Select Fall Delay; 12\20\36-bit words	0.95\ 0.94\ 0.95	2.39\ 2.56\ 2.78	0.02\ 0.02\ 0.02
Word-Match Rise Delay; 12\20\36-bit words	0.93\ 0.92\ 0.93	4.30\ 5.50\ 7.68	0.02\ 0.02\ 0.03
Word-Match Fall Delay; 12\20\36-bit words	1.00\ 1.00\ 1.00	5.73\ 6.74\ 8.35	0.26\ 0.26\ 0.26
Write Delays; 12\20\36-bit words	1.00\ 1.00\ 1.00	11.35\11.78\13.30	0.23\ 0.24\0.23
Read Delays; 12\20\36-bit words	0.99\ 9.98\ 0.97	12.61\13.80\15.11	0.39\ 0.38\ 0.36
Search-All Delays 12\20\36-bit words	1.00\ 1.00\ 1.00	14.33\15.32\17.13	0.33\ 0.33\ 0.33

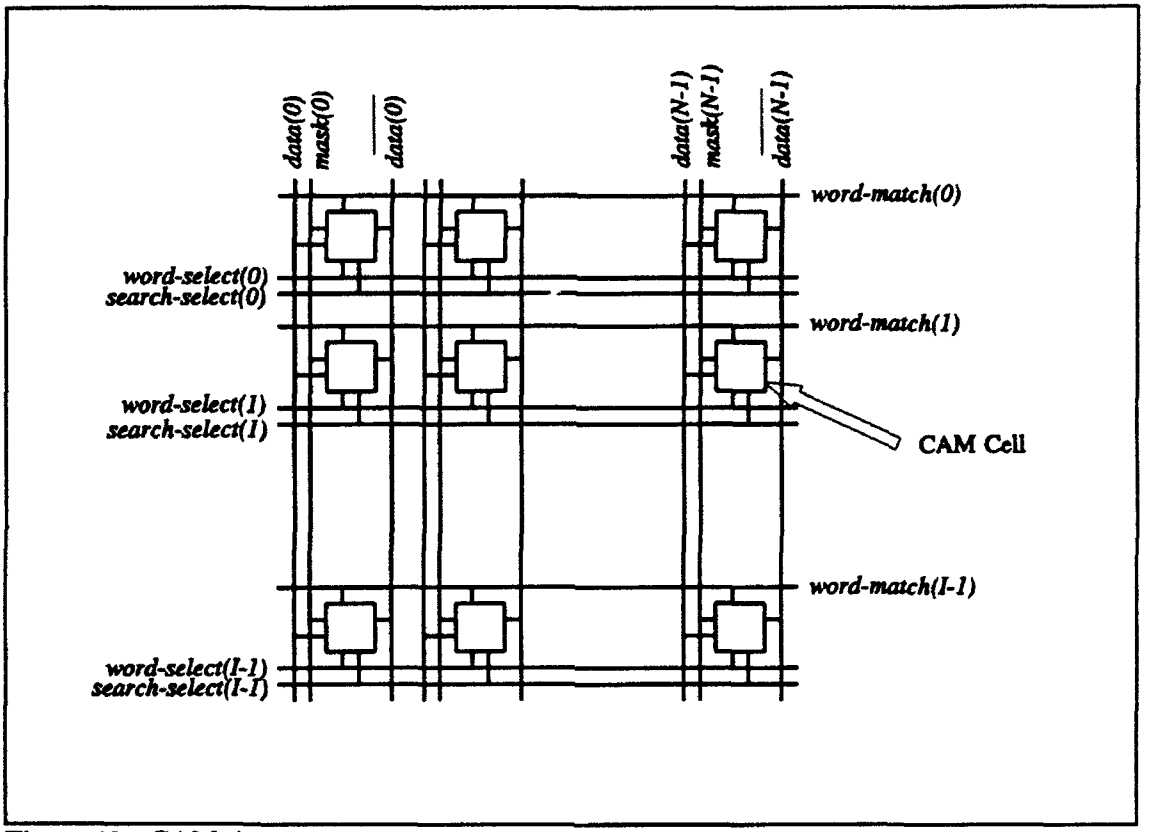

Figure 13. CAM Array.

search instruction for a CAM cell containing a 1 is similar.

4.3.3 Data-Read Circuit Description. The CAM array and the data-write circuit drive the data-read circuit using the *data* and $\overline{data}$ ports. From this stimuli, the data-read circuit forms the *N*-element *data-out* port. By changing the generic from *J* to *N*, the data-read circuit is the same as used by the SRAM. The CAM read rise and fall delays are defined the same as the SRAM read rise and fall delays except the CAM reference is *control-in(6)*. The transition from steady-state to a low value using external stimulus was recorded to be consistently less than 1.0 ns.

Table 20 gives the read delays. As the number of words in the CAM array increases, the precharge voltage range decreases which causes the read-circuit output-voltage range to decrease. The decreased output-voltage range decreases the noise margin and skews the

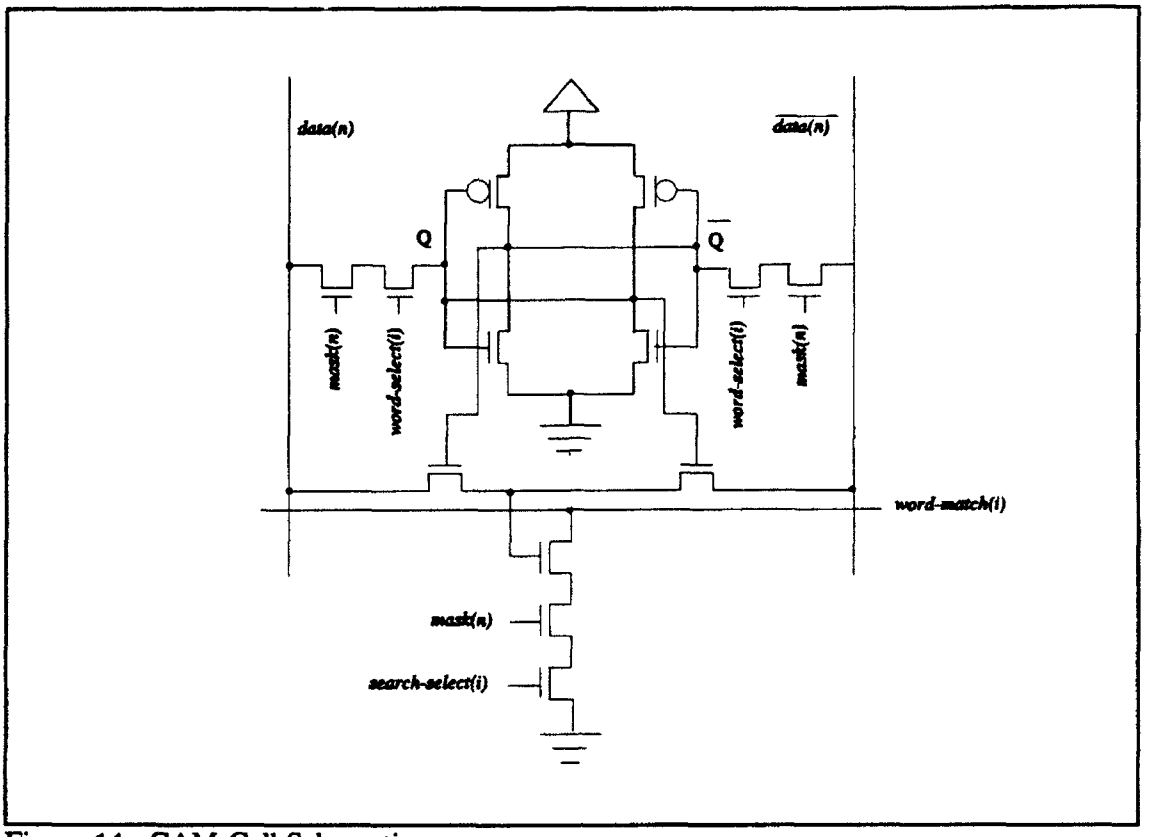

Figure 14. CAM-Cell Schematic.

Table 19

CAM Cell Transistor-Gate Dimensions.

CAM Cell Gate Length = 2λ	Gate Width (λ)
All P-Type Transistors	12
All N-Type Transistors	3

switching times. The result is attributed to the change in voltage range. The linear progression in the read delay is re-established with the redesigned precharge circuit discussed within the data-write circuit description.

Table 20

Read Delays for the CAM Architecture.

Read Circuit	read delay (ns) 12 bits reference: *control-in(6)* 50% pt	read delay (ns) 20 bits reference: *control-in(6)* 50% pt	read delay (ns) 36 bits reference: *control-in(6)* 50% pt
8 words	1.7 - 3.8 V t_r=10.8 t_f= 8.0	1.7 - 3.8 V t_r=11.2 t_f= 8.4	1.7 - 3.8 V t_r=12.2 t_f= 8.8
16 words	1.8 - 3.8 V t_r=15.0 t_f=12.1	1.8 - 3.8 V t_r=15.7 t_f=12.3	1.8 - 3.8 V t_r=17.1 t_f=12.5
32 words	1.9 - 3.7 V t_r=22.7 t_f=19.1	1.9 - 3.8 V t_r=24.1 t_f=18.7	2.0 - 3.8 V t_r=21.3 t_f=17.5
64 words	2.0 - 3.6 V t_r=33.6 t_f=21.4	2.1 - 3.6 V t_r=34.9 t_f=20.8	2.1 - 3.6 V t_r=36.2 t_f=21.0
128 words	2.3 - 3.2 V t_r=50.8 t_f=23.4	2.3 - 3.2 V t_r=51.3 t_f=23.3	2.3 - 3.2 V t_r=48.2 t_f=22.5

The rise delay for the 36-CAM cell by 32-word CAM-array size is shorter than the rise delay for the 20-CAM cell by 32-word CAM-array size. The rise delay for the 36-CAM cell by 120-word CAM-array size is shorter than either the 12-CAM cell or 20-CAM cell by 128-word CAM-array sizes. These three data points were remeasured with the same result, but no explanation for this counter-intuitive result is evident.

Table 18 provides the least-square analysis of the delays. As is seen from the table, the read rise-delay slopes are near constant for the three word lengths, and the offsets linearly increase with a linear increase in the word length. The read fall delays are not externally driven, and their non-linearity are attributed to the data and $\overline{data}$ precharge delays.

4.3.4 Word-Select Circuit Description. The word-select circuit performs two functions. First, it controls the execution of the write, read, and associative-search instructions. Second,

it stores the search results. The word-select circuit uses the *control-in* port stimulus, provided by the host, to execute its portion of an instruction. From the *control-in* port stimulus, it can stimulate either the *word-select*, or *search-select* port, or store the *word-match* port stimulus.

The word-select circuit consists of three subcircuits: the control-1 driver, the control-2 driver, and the phrase-select circuit. The control-1 and control-2 drivers perform a function similar to the SRAM driver. They accept the *control-in* port stimulus to form the *control* and $\overline{control}$ port stimuli used by the phrase-select circuit. The phrase-select circuit accepts the *control* and $\overline{control}$ port stimuli to execute the appropriate instruction.

Figure 15 shows the control-1 driver schematic, and Table 21 gives the transistor-gate dimensions. *Control-in(0)*, *(1)*, *(3)*, *(4)*, *(6)*, and *(7)* from the host are inputs to the control-1 drivers forming *control(0)*, *(1)*, *(3)*, *(4)*, *(6)*, and *(7)*. Figure 16 shows the control-2 driver schematic, and Table 22 gives the control-2 driver transistor-gate dimensions. The *control-in(2)*, and *control-in(5)* ports are inputs to the control-2 drivers forming the *control(2)*, *control(5)*, $\overline{control}(2)$, and $\overline{control}(5)$ ports.

The control-1 and control-2 inverters are cascaded so that each progressive inverter has n-type and p-type transistor widths two times as large as the previous inverter. The cascading is necessary to drive the 10 to 29 fF/word capacitance on the *control(k)* port.

The *control(2)* and $\overline{control}(2)$ ports placed the greatest capacitance load upon the drivers, so they were chosen for measurement. The control and $\overline{control}$ delays are referenced from the *control-in(2)* port. The control rise delay is the propagation delay necessary for the *control(2)* port to reach the 50% point in its transition from ground to V_{dd}. The control fall delay is the propagation delay necessary for the *control(2)* port to reach the 50% in its transition from V_{dd} to ground. The $\overline{control}$ rise delay is the propagation delay necessary for the $\overline{control}(2)$ port to reach the 50% point in its transition from ground to V_{dd}. The $\overline{control}$ fall delay is the propagation delay necessary for the $\overline{control}(2)$ port

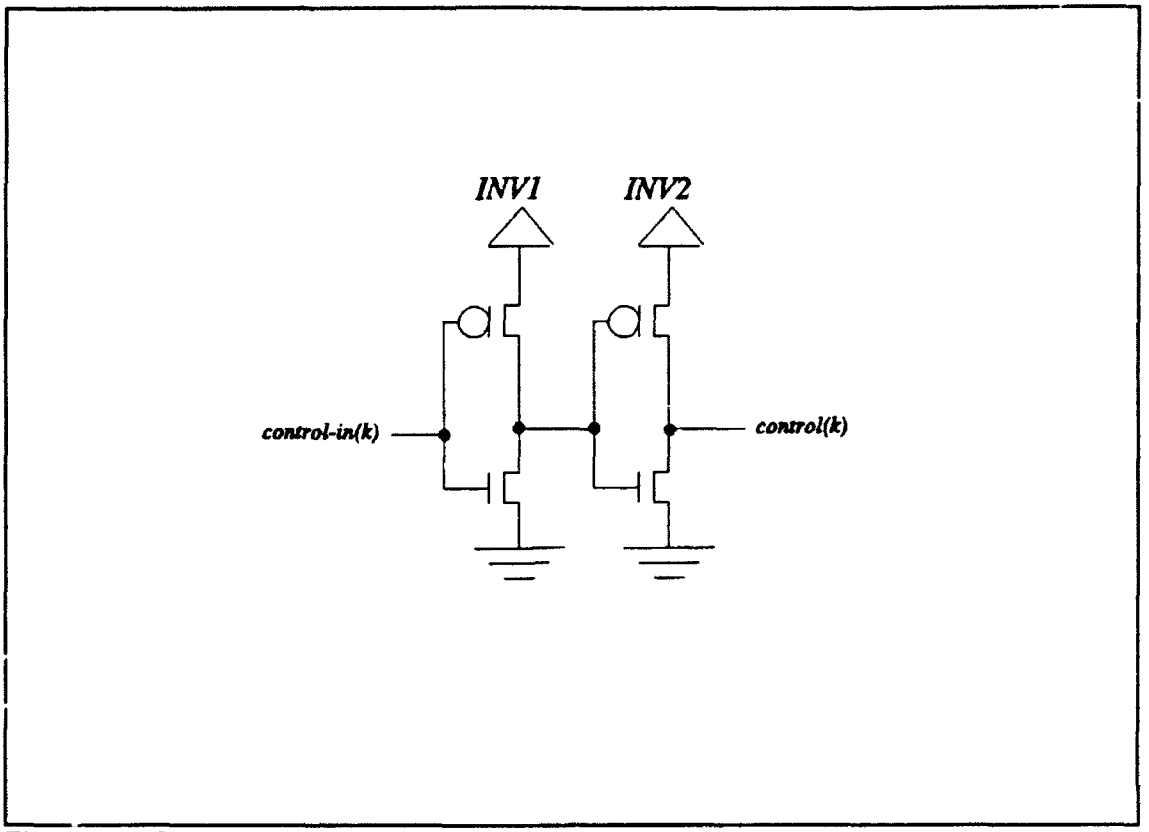

Figure 15. CAM Control-1 Driver Schematic.

Table 21

Control-1 Driver Transistor-Gate Dimensions.

Control-1 Driver Gate Length = 2λ	Gate Width (λ)
INV1 P	8
INV1 N	4
INV2 P	16
INV2 N	8

to reach the 50% point in its transition from V_{dd} to ground.

Table 23 gives the control, $\overline{control}$, and control(4) delays for simulated CAM-array sizes. Table 18 shows that the delays linearly increase with a linear increase in the number

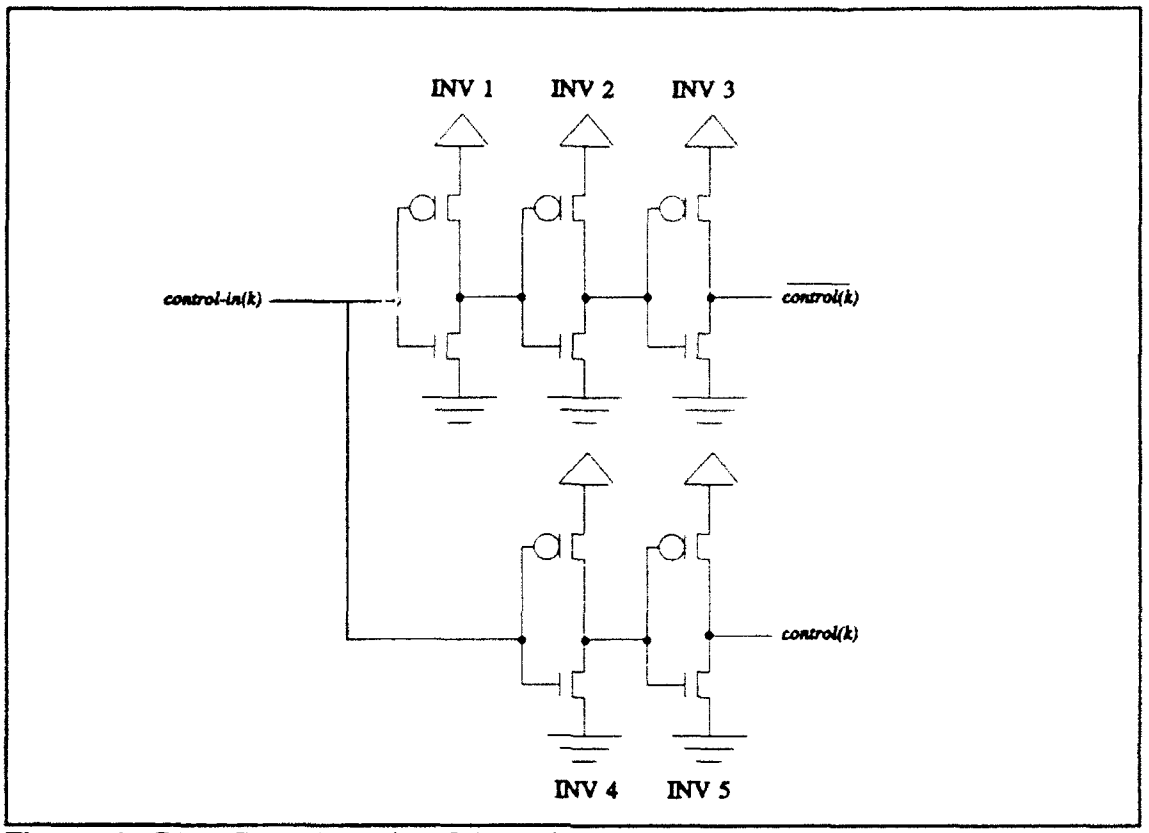

Figure 16. CAM Control-2 Driver Schematic.

of words. The load upon control(2) is slightly larger than the load upon $\overline{\text{control}}(2)$. This causes the control rise and fall-delay slopes to be slightly larger than the $\overline{\text{control}}$ rise and fall-delay slopes. The difference in offsets between the control and $\overline{\text{control}}$ ports are attributed to the difference in the number of inverters required to form the ports. The control(4) fall delay for a 32-word CAM array is smaller than the delay for the 64-word CAM array. This difference is attributed to the difference between circuit models. The first model has a dynamically changing load on control(4), while the second load is static.

The most complicated circuit in the CAM is the phrase-select circuit. It performs two functions. It selects the words involved in the read, write, and associative-search instructions, and it stores the results of previous instructions. The phrase-select circuit consists of four subcircuits. It requires a word circuit that selects the words involved in a read, write, or

Table 22

Control-2 Driver Transistor-Gate Dimensions.

Control-2 Driver Gate Length = 2λ	Gate Width (λ)
INV1 P	6
INV1 N	3
INV2 P	8
INV2 N	4
INV3 P	16
INV3 N	8
INV4 P	8
INV4 N	4
INV5 P	16
INV5 N	8

associative-search instruction. It requires word/search port drivers to select the *word-select* and *search-select* ports of the CAM array. It requires match circuitry to store the results of an associative-search instruction, and it requires a multiple-match resolution (MMR) circuit used to select one word of multiple-matching words for a write or read instruction.

4.3.4.1 Word Circuit and Word/Search Port Drivers Description. Each word within a phrase has a word circuit that can select one of four options to pass to the word/search line drivers. Figure 17 shows the word-circuit schematic. It consists of a 4-to-1 multiplexer, and an AND-gate. As show in Table 24, *control(3)*, and *control(4)* determine which multiplexer option is selected.

The AND-gate has two inputs, the output of the 4-to-1 multiplexer and *S19*. When the match circuit contains a matching result from a previous search, *S19* is enabled; otherwise, it is disabled. Therefore, when the results of a previous search yield a match, the 4-to-1

Table 23

Control, $\overline{\text{Control}}$, and Control(4) Delays for the CAM Architecture.

PE Array	control delay (ns) reference: *control-in(2)* 50% pt range: 0 to 5 V	$\overline{\text{control}}$ delay (ns) reference: *control-in(2)* 50% pt range: 0 to 5 V	control(4) delay (ns) reference: *control-in(4)* 50% pt range: 0 to 5 V
8 words	t_r= 1.4 t_f= 1.0	t_r= 1.5 t_f= 1.4	t_r= 1.2 t_f= 0.9
16 words	t_r= 2.3 t_f= 1.5	t_r= 2.4 t_f= 1.8	t_r= 3.3 t_f= 2.5
32 words	t_r= 4.3 t_f= 2.4	t_r= 4.2 t_f= 2.6	t_r= 7.8 t_f= 3.1
64 words	t_r=10.6 t_f= 5.3	t_r=10.0 t_f= 5.4	t_r=12.0 t_f=18.3
128 words	t_r=19.8 t_f= 8.9	t_r=18.8 t_f= 9.5	t_r=24.4 t_f=37.3

multiplexer selection is passed to the word/search line driver circuitry.

Figure 17 also shows the word/search port driver schematic. It consists of two NAND-gates along with six inverters per word. The word/search line drivers perform two functions. First, they use *control(5)* and $\overline{control(5)}$ to select whether the instruction is a read/write or a search. When *control(5)* is enabled, either a read or write instruction is selected. When it is disabled, an associative-search instruction is selected. Second, *control(6)* is used to initiate the *word-select(i)* or *search-select(i)* port stimulation. When *control(6)* is enabled, either the *word-select(i)* or *search-select(i)* port is stimulated.

The word-select, search-select and word-match delays are referenced from the *control-in(6)* port transition. The word-select/search-select rise delay is the propagation delay necessary for the *word-select(i)/search-select(i)* port to reach the 50% point in its transition from ground to V_{dd}. The word-select/search-select fall delay is the propagation delay necessary for the *word-select(i)/search-select(i)* port to reach the 50% point in its transition from V_{dd} to ground. The word-match rise delay is the propagation delay necessary for the

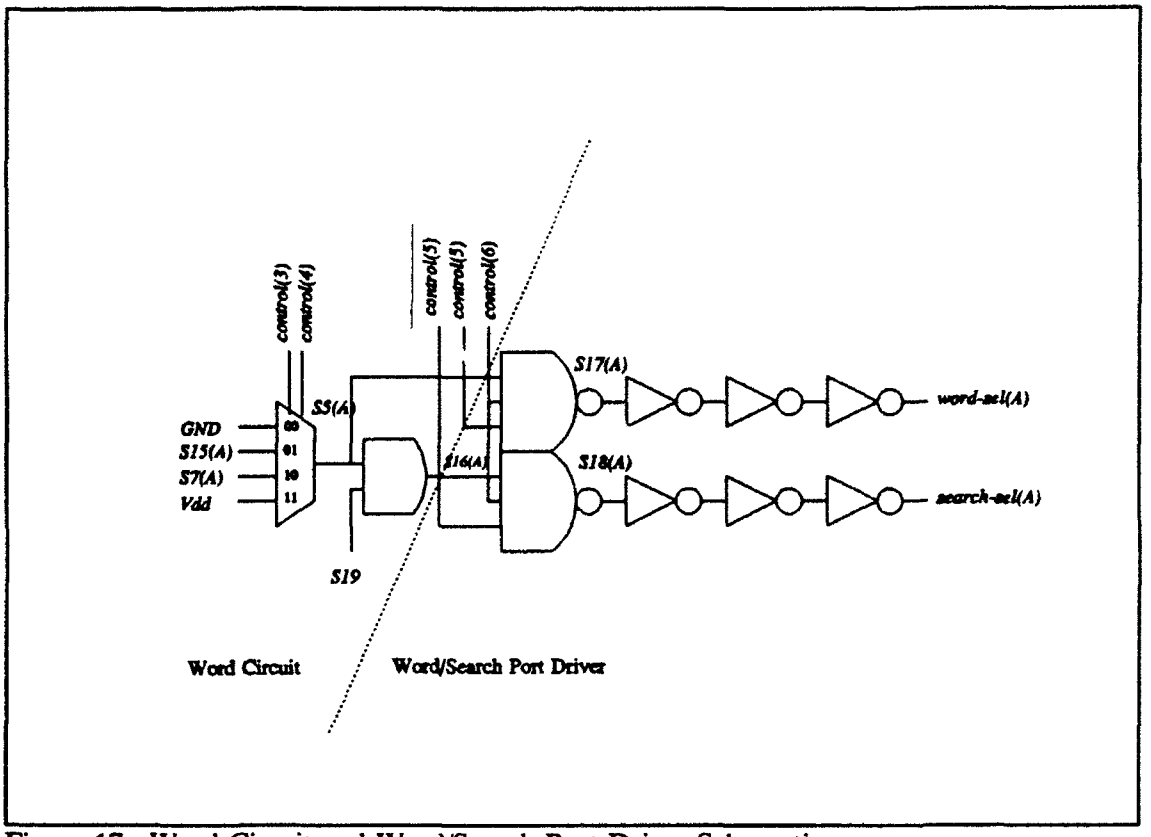

Figure 17. Word Circuit and Word/Search Port Driver Schematic.

word-match(i) port to reach the 50% point in its transition from a steady-state voltage of approximately 1.4 V to approximately 3.4 V, and the word-match fall delay is the propagation delay necessary for the *word-match(i)* port to reach the 50% point in its transition from approximately 3.4 V to steady state.

Tables 25, 26, and 27 give the delays to drive the *word-select(i)*, *search-select(i)*, and *word-match(i)* ports for simulated CAM-array lengths. As can be seen from the tables, the delay to initiate a read/write/search instruction and to drive the appropriate words in the CAM array linearly increases with a linear increase in the number of words.

Table 18 shows that the word-select rise delay, search-select rise delay, and word-match fall delay slopes are approximately equal. Likewise, the word-select fall delay, search-select fall delay, and word-match rise delay slopes are approximately equal. The word-match

Table 24

Control(3) and Control(4) Port Function.

Control(3)	Control(4)	Option
0	0	GND is passed to AND-gate
0	1	Output of MMR is passed to AND-gate.
1	0	Contents of match circuit is passed to AND-gate.
1	1	Vdd is passed to AND-gate.

port fall transition is activated by the search-select port transition from ground to V_{dd}, so its offset should be larger.

4.3.4.2 Match-Circuit Description. Figure 18 shows the match-circuit schematic. The match circuit accepts and stores the results of an associative-search instruction. Once stored, the results are used to form *control-out* (*control(4)* for the next phrase-select circuit) and are used as an input to the multiple-match resolution function. *Search-select(A)*, *control-in(0)*, *(1)*, *(2)*, *(3)*, and *(4)* determine the circuit operation.

The state of *search-select(A)* ensures that word matches are stored only for words searched against. If a word is selected for an associative-search instruction, then *search-select(A)* will be enabled and the word-match results can be passed through AND-gate #1; otherwise, the 0 output of AND-gate #1 will be passed to mux2x1#2.

Control(0) along with *search-select(A)* controls mux2x1#1. When *control(0)* is disabled and *search-select(A)* is enabled, the results of the search are passed to mux2x1#2. This is an equal-to operation. On the other hand, if *control(0)* is enabled, the inverse of the results is passed to mux2x1#2. This is a not-equal-to operation.

Table 25

Word-Select Delays for the CAM Architecture.

CAM Array	word-select delay (ns) 12 bits reference: *control-in(6)* 50% pt range: 0 to 5 V	word-select delay (ns) 20 bits reference: *control-in(6)* 50% pt range: 0 to 5 V	word-select delay (ns) 36 bits reference: *control-in(6)* 50% pt range: 0 to 5 V
8 word	t_r= 6.7 t_f= 2.7	t_r= 7.0 t_f= 3.0	t_r= 7.8 t_f= 3.4
16 word	t_r= 8.5 t_f= 3.2	t_r= 8.9 t_f= 3.4	t_r= 9.7 t_f= 3.8
32 word	t_r=10.9 t_f= 4.0	t_r=11.3 t_f= 4.2	t_r=12.1 t_f= 4.7
64 word	t_r=17.1 t_f= 6.2	t_r=17.5 t_f= 6.4	t_r=18.5 t_f= 6.8
128 word	t_r=27.8 t_f= 9.7	t_r=28.2 t_f= 9.9	t_r=29.0 t_f=10.3

Control(1) controls mux2x1#2. When *control(1)* is enabled, *word-match(A)* is passed; otherwise, $S5(A)^*S7(A)$ is passed. $S5(A)^*S7(A)$ resets words during a read instruction. If after reading a matching word the next word is desired, $S5(A)^*S7(A)$ will reset the word just read allowing the MMR to select the next matching word.

Control(2) controls a master-slave flip-flop. When *control(2)* is enabled, the master portion of the master-slave flip-flop accepts data; otherwise, the slave portion accepts data. During an associative-search instruction *control(2)* is enabled to store the search results.

Control(3) and *(4)* along with the search results determine *control-out*. When *control(3)* and *(4)* are disabled, *control-out* is disabled. When *control(3)* is disabled and *control(4)* is enabled, the results of the previous search are used to form *control-out*. If any match occurred, then *control-out* is disabled; otherwise, *control-out* is enabled. *Control(7)* is used as a reset line for initialization. When *control(3)* is enabled and *control(4)* is disabled, *control-out* is disabled. When *control(3)* and *(4)* are enabled, *control-out* is enabled.

Table 26

Search-Select Delays for the CAM Architecture.

CAM Array	search-select delay (ns) 12 bits reference: *control-in(6)* 50% pt range: 0 to 5 V	search-select delay (ns) 20 bits reference: *control-in(6)* 50% pt range: 0 to 5 V	search-select delay (ns) 36 bits reference: *control-in(6)* 50% pt range: 0 to 5 V
8 word	t_r= 7.4 t_f= 2.3	t_r= 7.6 t_f= 2.5	t_r= 8.0 t_f= 2.7
16 word	t_r= 9.4 t_f= 2.7	t_r= 9.5 t_f= 2.8	t_r=10.0 t_f= 3.0
32 word	t_r=11.6 t_f= 3.2	t_r=11.8 t_f= 3.4	t_r=12.3 t_f= 3.6
64 word	t_r=18.0 t_f= 4.6	t_r=18.2 t_f= 4.7	t_r=18.6 t_f= 4.9
128 word	t_r=28.5 t_f= 6.3	t_r=28.8 t_f= 6.5	t_r=29.1 t_f= 6.6

4.3.4.3 MMR-Circuit Description. The MMR-circuit schematic is shown in Figure 19. It is used to select a single word from a subset of matching words for write and read instructions. The MMR circuit takes the search results and selects the first matching result to pass to *S15*. All other matching words are suppressed, so *S15* will have at most a single enabled element [19].

4.4 CAM Operation Description

The CAM can operate in one of two modes, word or phrase. It can execute one of eight instructions in each mode: write-all, write-subset, write, read, search-all-equal, search-all-not-equal, search-subset-equal, search-subset-not-equal. The process necessary to execute each instruction is discussed next.

4.4.1 Word Operation Mode. In the word mode, the CAM can perform a variety of write, read, and associative-search instructions, with each requiring three phases. These instructions are presented next. First, the write instructions are presented. Second, the read

Table 27

Word-Match Delays for the CAM Architecture.

CAM Array	word-match (ns) 12 bits reference: *control-in(6)* 50% pt range: 1.4 to 3.4 V	word-match (ns) 20 bits reference: *control-in(6)* 50% pt range: 1.4 to 3.2 V	word-match (ns) 36 bits reference: *control-in(6)* 50% pt range: 1.4 to 3.2 V
8 word	t_r= 4.2 t_f= 8.3	t_r= 5.3 t_f= 9.4	t_r= 7.5 t_f=11.1
16 word	t_r= 4.5 t_f=10.4	t_r= 5.8 t_f=11.3	t_r= 8.0 t_f=13.1
32 word	t_r= 5.4 t_f=12.6	t_r= 6.7 t_f=13.6	t_r= 9.1 t_f=15.5
64 word	t_r= 6.9 t_f=19.0	t_r= 8.3 t_f=20.0	t_r=11.6 t_f=21.8
128 word	t_r= 8.9 t_f=29.4	t_r=10.4 t_f=30.5	t_r=13.2 t_f=32.2

instruction is presented, and third, the associative-search instructions are presented.

4.4.1.1 Write Instructions. The host can select one of three write instructions: write-all, write-subset, and write. All three write instructions begin when the host places data onto the *data-in* port, identifies the bit-slice involved in the write instruction using the *mask-in* port, enables the *write* port, and selects the write instruction using the *control-in* port, without enabling *control-in(6)*. Once sufficient time has passed to form *data*, $\overline{data}$, and *mask*, and to distribute the *control-in* stimulus throughout the word-select circuitry, the host directs the word-select circuit to enable *control-in(6)*. Once the data are written into the CAM array, the host directs the word-select circuit to disable *control-in(6)*, then the *data-in*, *mask-in*, and *write* port stimuli can change.

The write-all instruction writes the data specified by the *data-in* and *mask-in* port stimuli to every word in a CAM array not exceeding 32 words. One use for this instruction is to disable the valid-data field of the CAM. During the first phase of the write-all

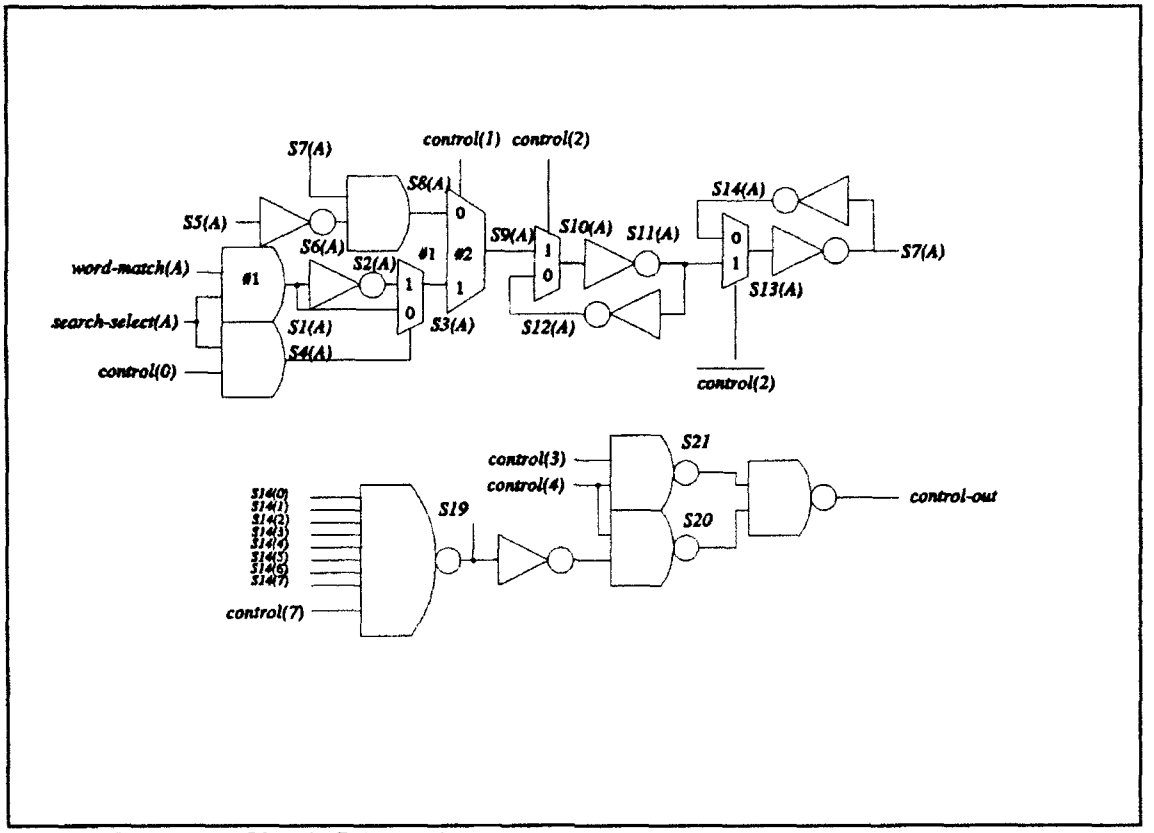

Figure 18. Match-Circuit Schematic.

instruction, the host provides to the word-select circuit *control-in* port the stimulus 00011100. During the second phase, the word-select circuit receives 00011110, and during the third phase, it receives 00011100. *Control-in(2)* remains disabled throughout the three phases, so no match information is stored. Since *control-in(3)* and *control-in(4)* remain enabled throughout the three phases, a 1 is passed to *S5(A)* for all *A*. *Control-in(5)* remains enabled, so the *search-select* port cannot be enabled. When *control-in(6)* is enabled during the second phase, the *word-select* port is enabled, and when it is disabled during the third phase, the *word-select* port is disabled. When enabled, every element of *word-select* is enabled.

The write-subset instruction writes the data specified by the *data-in* and *mask-in* stimuli to a subset of words in the CAM array. This instruction uses the results of a previous associative-search instruction to write data into a subset of words not exceeding 32. During

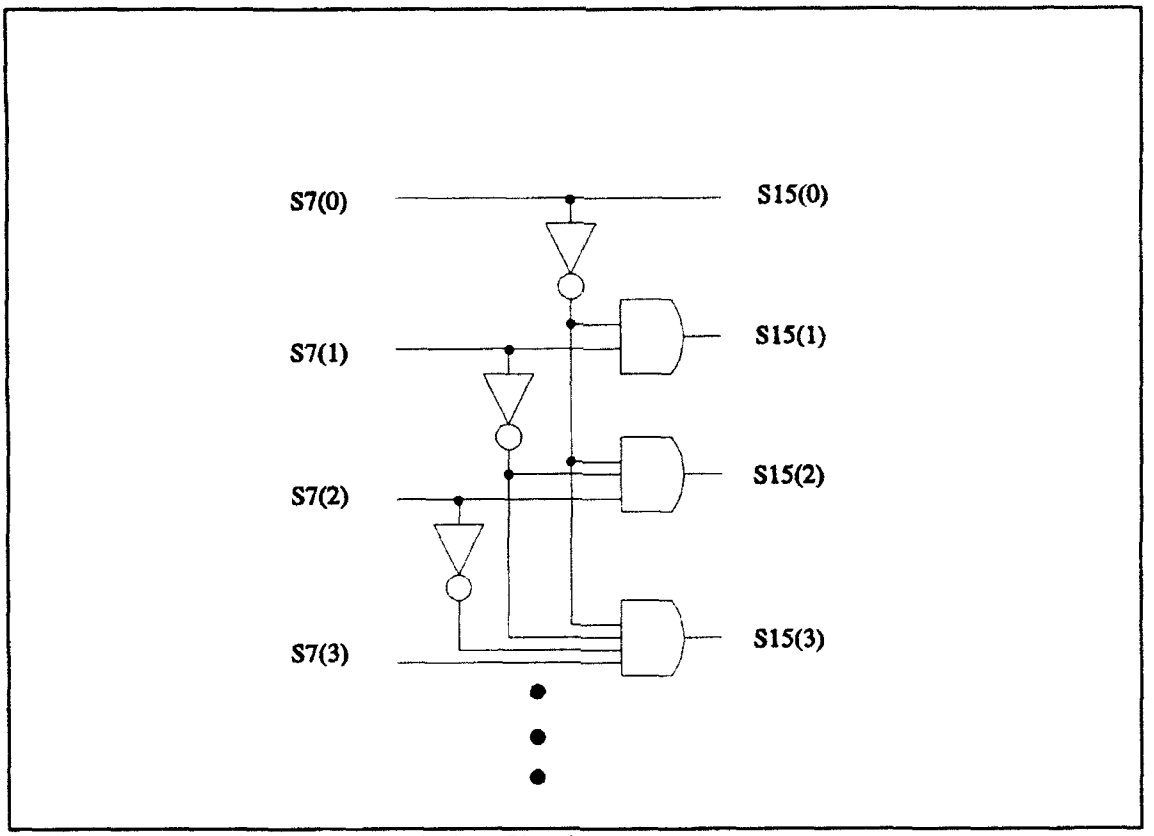

Figure 19. Multiple-Match-Resolution Circuit Schematic.

the three phases of the write-subset instruction, the host provides to the word-select circuit *control-in* port the stimuli 00010100, 00010110, and 00010100, respectively. The write-subset instruction operates similarly to the write-all instruction, except when *control-in(6)* is enabled during the second phase, *S7* is passed to *word-select*.

The write instruction writes the data specified by the *data-in* and *mask-in* port stimuli to a single word in the CAM array. During the three phases of the write instruction, the host provides to the word-select circuit the stimuli 00001101, 00101111, and 00001101, respectively. The transition of *control-in(2)* from 0 to 1 and back to 0 causes the de-selection of the written word from further write instructions that may occur before the next associative-search instruction. In this instruction, when *control-in(6)* is enabled during the second phase, *S15* is passed to *word-select*.

Of the three write instructions, write-all is the most time consuming because the CAM writes to every word in the CAM array. The time to execute a *write-all instruction* is tabulated in Table 28. The first number in each cell of the table represents the first and second phases of the instruction. The first and second phases can overlap because the delay to form *data*, $\overline{data}$, and *mask* is much shorter than the delay to select the word. A 3.6 ns delay is added to allow the data to be stored. The third number in each cell represents the third phase. The third phase includes the delay to disable the *word-select* port. The 64 and 128-word array write-all times were calculated with the assumption that the data-write circuit is redesigned to properly switch the CAM cell contents. As is seen from Table 28, the write delays linearly increase as the number of words in the CAM array linearly increases.

From Table 18, notice that increasing the number of CAM cells in a word increases the offset, but the slopes are nearly constant. From the least-square analysis of the three offsets, a single equation characterizing the w ite-all instruction is formed. Equation (48) approximates the CAM write-all instruction execution time.

$$\tau_{write\ all} = (10.325 + 0.079(T(s) + T_{wdf} + T_{pf}) + 0.233I)\ ns \qquad (48)$$

The percentage difference between the equation and the tabulated write-all times is measured to be less than 3.1% for all simulated CAM-array sizes. The percentage difference was calculated by dividing the difference between the measured and calculated delay by the measured delay.

4.4.1.2 Read Instruction. The host can select only one read instruction. The read instruction reads data from a single word, which is identified during a previous search. The instruction begins when the host identifies the bit-slices involved using *mask-in*, and selects the instruction using *control-in*, without enabling *control-in(6)*. Once sufficient time has passed to form *mask*, and to distribute the *control-in* stimulus throughout the word-select

circuit, the host directs the word-select circuit to enable *control-in(6)*. Once the data are read, the host directs the word-select circuit to disable *control-in(6)* elements, then the *mask-in* stimulus can change. During the three phases of the read instruction, the host provides to the word-select circuit the stimuli 00001101, 00101111, 00001101, respectively. When *control-in(6)* is enabled during the second phase, *S15* is passed to *word-select*.

Table 28

Write-All Time for the CAM Architecture.

Write-All	12 bits (ns)	20 bits (ns)	36 bits (ns)
8 words	6.7 + 3.6 + 2.7 = 13.0	7.0 + 3.6 + 3.0 = 13.6	7.8 + 3.6 + 3.4 = 14.8
16 words	8.5 + 3.6 + 3.2 = 15.3	8.9 + 3.6 + 3.4 = 15.9	9.7 + 3.6 + 3.8 = 17.1
32 words	10.9+ 3.6 + 4.0 = 18.5	11.3+ 3.6 + 4.2 = 19.1	12.1+ 3.6 + 4.7= 20.4
64 words	17.1+ 3.6 + 6.2 = 26.9	17.5+ 3.6 + 6.4 = 27.5	18.5+ 3.6 + 6.8= 28.9
128 words	28.0+ 3.6 + 9.7 = 41.3	28.2+ 3.6 + 9.9 = 41.7	29.0+ 3.6 + 10.3= 42.9

The time to execute a read instruction is tabulated in Table 29. The first number in each cell of the table again represents the first and second phases of the instruction. The first and second phases can overlap because the delay to form *data*, $\overline{data}$, and *mask* is much shorter than the delay to select and read a word. The third number in each cell of the table represents the third phase, which includes the delay to disable the *word-select* port. The data-precharge delay is not included for the same reasons as discussed in Section 3.4. As is seen from Table 29, with one exception, the read delays linearly increase as the number of words in the CAM array linearly increases. The single exception occurs when transitioning from

the 20-bit by 128-word CAM-array size to the 36-bit by 128-word CAM-array size. Section 4.3.3 explains the cause of this discrepancy.

Table 18 provides the least-square analysis. Similar to the write delay, the read delay has a near constant slope for the three word lengths, and as the number of CAM cells within a word increases so does the offset. From a least-square analysis of the three offsets a single equation characterizing the read instruction is formed. Equation (49) approximates the CAM read instruction execution time.

$$\tau_{read} = (11.552 + 0.101(T(s) + T_{wdf} + T_{pp}) + 0.376l)\ ns \qquad (49)$$

The percentage difference between the equation and the tabulated read times is measured to be less than 16.8% for all simulated CAM-array sizes and less than 9.2% for CAM-array lengths greater than eight words. The increased error as compared to the write instruction is attributed to the data-precharge voltage degradation that occurs for CAM-array lengths greater than 64 words.

4.4.1.3 Associative-Search Instructions. The host can select one of four BPI, RCI instructions: search-all-equal, search-all-not-equal, search-subset-equal, and search-subset-not-equal. All four instructions begin when the host places the comparand onto the *data-in* port, identifies the bit-slices involved in the instruction using the *mask-in* port, enables the *write* port, and selects the instruction using the *control-in* port, without enabling *control-in(6)*. Once sufficient time has passed to form *data*, $\overline{data}$, and *mask*, and to distribute the *control-in* stimulus throughout the word-select circuitry, the host directs the word-select circuit to enable *control-in(6)*. Once the comparison results are stored in the word-select circuit, the host directs the word-select circuit to disable *control-in(6)*, then the *data-in*, *mask-in*, and *write* port stimuli can change.

Table 29

Read Time for the CAM Architecture.

Read	12 bits (ns)	20 bits (ns)	36 bits (ns)
8 words	10.8 + 2.7 = 13.5	11.2 + 3.0 = 14.2	12.2 + 3.4 = 15.6
16 words	15.0 + 3.2 = 18.2	15.7 + 3.4 = 19.1	17.1 + 3.8 = 20.9
32 words	22.7 + 4.0 = 26.7	24.3 + 4.2 = 28.5	21.3 + 4.7 = 26.0
64 words	33.6 + 6.2 = 39.8	34.9 + 6.4 = 41.3	36.2 + 6.8 = 43.0
128 words	50.8 + 9.7 = 60.5	51.3 + 9.9 = 61.2	48.2 +10.3 = 58.5

The search-all-equal instruction compares the data specified by *data-in* and *mask-in* to every word in the CAM array. This operation is designed to search the valid-data field of the CAM array to determine all the words that are available for further operations, but can be used to search any field in the CAM. The search-all-equal instruction requires three phases with a *control-in* stimulus of 01011000, 01111010, and 01011000. *Control-in(0)* remains disabled during the three phases, so the search results are passed into the word-select circuit for future storage. The enabling of *control-in(2)* during the second phase allows the match results to be stored in the master portion of the master-slave flip-flop, *S11*. The disabling of *control-in(2)* during the third phase allows the match results to be stored in the slave portion of the master-slave flip-flop, *S7*. *Control-in(5)* remains disabled, so the *word-select* port cannot be enabled. When *control-in(6)* is enabled during the second phase, every element in the *search-select* port is enabled.

The search-all-not-equal instruction performs the inverse of search-all-equal. It is executed in the same fashion as the search-all-equal instruction with one exception. *Control-in(0)* is enabled for the entire instruction.

The search-subset-equal instruction compares the data specified by *data-in* and *mask-in* to a subset of words in the CAM array. This instruction is designed to use the results of a previous search to select words to further search. The search-subset-equal instruction requires three phases, with a *control-in* stimuli of 01010000, 01110010, and 01010000, respectively. The search-subset-not-equal performs the inverse of search-all-equal. It is executed in the same fashion as search-subset-equal with one exception. *Control-in(0)* is enabled for the entire instruction.

The longest delays to search the CAM array occur when the CAM compares a single CAM cells of each word in the array. The word-match delays of Table 27 were obtained under this condition. The time to search the CAM is calculated by summing the word-match fall delay, the delay to transfer match results into the word-select circuit, the delay to change the control instruction to initiate the third phase, and the delay to form the MMR result. The appropriate delays are summed and provided in Table 30. As is seen from Table 30, the associative-search instruction execution time linearly increases as the number of words in the CAM array linearly increases.

From Table 18, notice that increasing the number of CAM cells in a word increases the offset, but the slopes are constant. From a least-square analysis of the three offsets, a single equation characterizing the search-all instruction is formed. Equation (50) approximates the CAM search-all instruction.

$$\tau_{cam}=(12.966+0.116(T(s)+T_{wdj}+T_{pf})+0.331l)\ ns \qquad (50)$$

The percentage difference between the equation and the tabulated search-all times is measured to be less than 4.5% for all simulated CAM-array sizes.

4.4.2 Phrase Operation Mode. The phrase mode is designed for real-time comparison of data to a fixed or a slowly changing database. In the phrase mode, data are written into

the CAM array in an orderly fashion, unlike the word operation mode. The reason for this restriction is to maintain a logical relationship between words in a phrase. A database can be placed into the CAM phrase by phrase, which means that data are written into the first word of the first phrase, then the second word of the first phrase, etc. Once completed with writing the first phrase, data are written into the second phrase.

Table 30

Search Time for the CAM Architecture.

Search-All	12 bits (ns)	20 bits (ns)	36 bits (ns)
8 words	8.3+3.7+1.5+3.4 = 16.9	9.4+3.7+1.5+3.4 = 18.0	11.1+3.7+1.5+3.4 = 19.7
16 words	10.4+ 3.7+2.4+3.4 = 19.9	11.3+3.7+2.4+3.4 = 20.8	13.1+3.7+2.4+3.4 = 22.6
32 words	12.6+3.7+4.2+3.4 = 23.9	13.6+3.7+4.2+3.4 = 24.9	15.5+3.7+4.2+3.4 = 26.8
64 words	19.0+3.7+10.6+3.4 = 36.7	20.0+3.7+10.6+3.4 = 37.7	21.8+3.7+10.6+3.4 = 39.5
128 words	29.4+3.7+19.8+3.4 = 56.3	30.5+3.7+19.8+3.4 = 57.4	32.2+3.7+19.8+3.4 = 59.1

Before the execution of write and read instructions and during the execution of associative-search instructions, the phrase and valid-data fields are searched to identify all valid words with a particular word location that are available for writing, reading, or searching. The results of a word search within a phrase are stored and can be used to select other word searches within the phrase. The results of the phrase search can be used to read matching records from the CAM array.

4.5 Comparison of the CAM to Other Similar Designs.

Though the research is a comparison of different organizations using a common design technology, it is important not to skew the results by comparing a poorly designed CAM to another organization that is well designed. The most comparable CAM designs are presented by Jones, Kadota, and Ogura [4, 7, 8].

Jones presented the single-chip array processor, which consists of 256 words with each word containing 37 bits [8]. It uses 2 μm minimum-feature width design rules with two layers of metal and requires 75 mm^2 die size (75 Mλ^2). It uses four 25 ns phases to execute an instruction. His design could not handle multiple-match resolution issues.

Kadota presented a content-addressable and reentrant memory which consists of 256 words with each word containing 32 bits [7]. It uses a 2 μm minimum-feature width design rule with two layers of metal, and requires 34 mm^2 die size (34 Mλ^2). It uses a 100 ns cycle to execute an instruction consisting of four 25 ns phases.

Ogura presented an associative memory consisting of 128 words with each word consisting of 32 bits [4]. It uses a 3 μm minimum-feature width design rule and uses double aluminum and requires 86.5 mm^2 die size (38.4 Mλ^2). It requires a 200 ns average instruction cycle with a 140 ns minimum instruction cycle.

As can be seen from Table 30, the CAM designed for this research endeavor compares well with similar designs. The CAM designed for this research requires approximately 100 ns to execute a search on 256 36-bit words, and requires approximately 16.27 mm^2 of a die (45.2 Mλ^2)

4.6 Summary and Conclusion

This chapter presented the CAM. It discussed the purpose of the design, presented a general organization, and gave the specific architectural implementation. It discussed how

to execute write, read, and associative-search instructions and through simulations calculated the time required to execute each instruction.

The CAM organization could have been less complicated and faster if the design were to execute only bit-position independent operations on an entire word, but the research required a CAM with features to allow the execution of bit-position dependent searches. The necessary features include the ability to mask bit-slices from being involved in a search, the ability to select a subset of words to search, the ability to identify all words that either match or do not match, and the ability to quickly identify whether at least one match exists. The architecture was designed with all these features.

V. Bit-Serial Word-Parallel Associative Memory

5.1 Introduction

This chapter presents the bit-serial word-parallel associative-memory (BSWPAM) organization. It was designed to demonstrate the relationship between BPD, RCI operations and an associative-memory organization that can compare a subset of bit-slices from the memory array to the corresponding subset of bits from the comparand. Three architecture representations of the organization were designed: the single, two, and four BSWPAM architectures. Each architecture executes the same set of associative-search instructions in one of two modes: word or phrase. Due to these similarities, these architectures are discussed in a single chapter.

The chapter develops the characteristic equations to approximate the write, read, and associative-search instruction-set execution times. To develop the background necessary to formulate the characteristic equations, the chapter begins by describing the organization and its general operation. It describes each architecture to the transistor level, and explains the procedures to execute the instruction set. It gives the instruction-set execution-time results for a variety of memory-array sizes. From these results, the characteristic equations are calculated.

For a VHDL description of each architecture, refer to Appendices D, E, and F.

5.2 BSWPAM Organization and General Operation Description

Figure 20 shows the BSWPAM organization. It consists of a memory array called the bit-serial array, a comparand register, a word-address selector, a bit-slice address selector, and a search-results register.

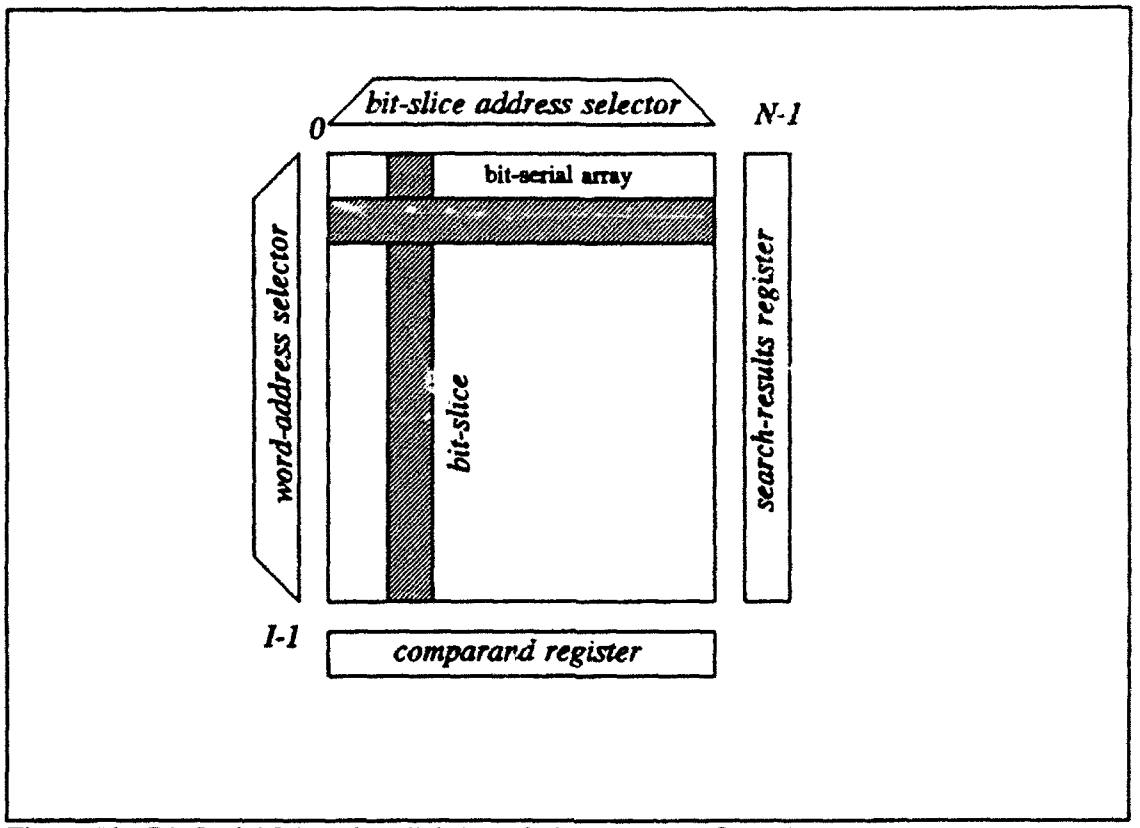

Figure 20. Bit-Serial Word-Parallel Associative-Memory Organization.

Data are stored into and retrieved from the bit-serial array one word at a time. The location of data storage and retrieval is defined by the word-address selector. Data can also be retrieved from the bit-serial array in sets of bit-slices, using the bit-slice address selector. The single BSWPAM architecture can access a single bit-slice of the bit-serial array at a time. The two BSWPAM architecture can access two consecutive bit-slices of the bit-serial array at a time, and the four BSWPAM architecture can access four consecutive bit-slices of the bit-serial array at a time. The bit-slice comparison results are stored in the search-results register.

To execute an associative-search operation using the single BSWPAM architecture, the current results of comparing a bit of the comparand to the corresponding bit-slice of the bit-serial array are stored in the search-results register and can be logically combined with

future bit-slice comparison results to eventually form the final result. Likewise, to execute an associative-search operation using the two BSWPAM architecture, the current results of simultaneously comparing two bits of the comparand and the corresponding two bit-slices of the bit-serial array are stored in the search-results register and combined to eventually form a final result. As expected, the four BSWPAM architecture simultaneously compares four bit-slices of the bit-serial array at a time.

The BSWPAM organization divides words into fields in the same manner as the CAM organization. It uses a single-bit valid-data field in both the word and phrase modes, and a three-bit phrase field in the phrase mode. Data are stored within a field using a magnitude number representation, so the first bit of a field represents the most significant bit of the data. Equality and inequality operations can start anywhere within the field because they are bit-position independent, but all other associative-search operations must start by first comparing the most significant bit position and storing the results, then by comparing the second most significant bit position, etc.

5.3 BSWPAM-Architecture Description

This section describes each BSWPAM architecture. Figure 21 shows the single BSWPAM architecture. It consists of ten components: the data-write circuit, the address driver, the slice-address driver, the address-decode circuit, the slice address-decode circuit, the bit-serial array consisting of I times N bit-serial cells, the data-read circuit, the slice data-read circuit, the processing-element (PE) array, and the address encoder. The two BSWPAM architecture uses the same ten components as mentioned above and requires an additional slice data-read circuit to transfer the second bit-slice of the bit-serial array into the PE array. The four BSWPAM architecture requires three additional slice data-read circuits.

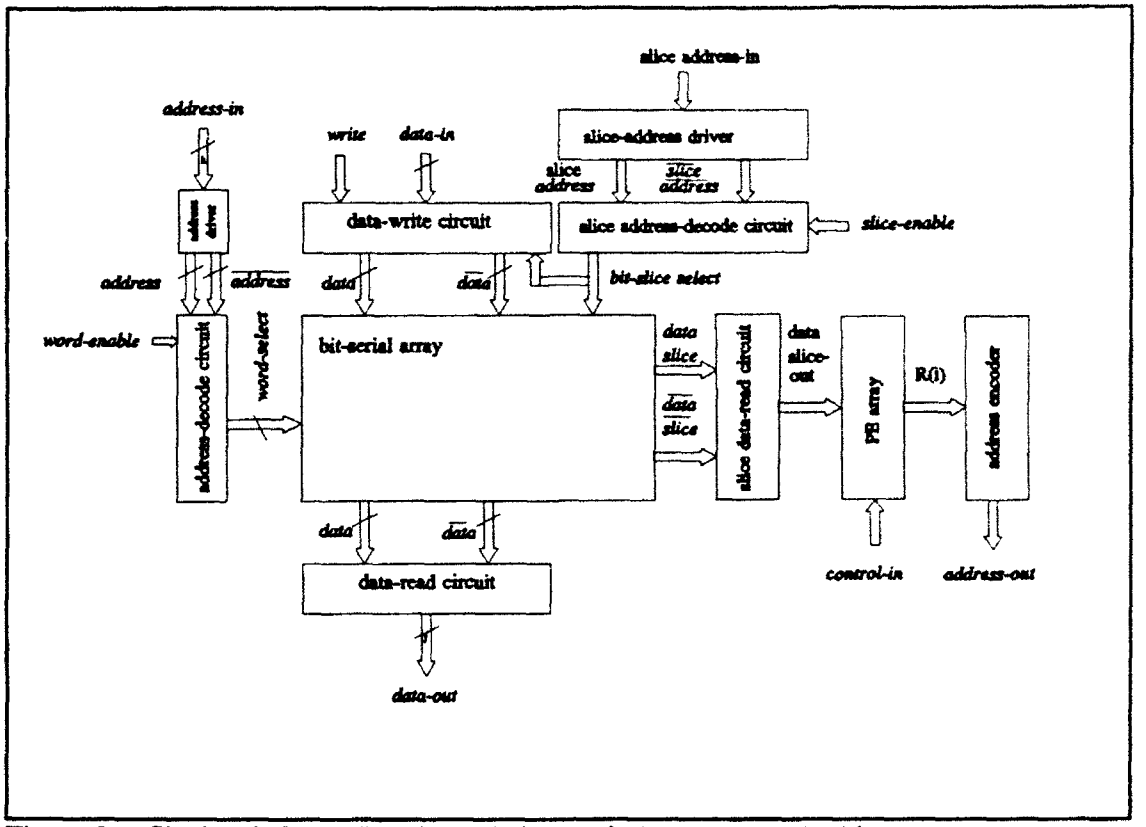

Figure 21. Single Bit-Serial Word-Parallel Associative-Memory Architecture.

Each architecture interfaces with the host using nine ports: *data-in*, *data-out*, *address-in*, *write*, *word-enable*, *control-in*, *slice-address-in*, *slice-enable*, and *address-out*. The host supplies data to each architecture using the N-element *data-in* port and retrieves data using the N-element *data-out* port. It selects the location of storage and retrieval using the K-element *address-in* port (where $K=\lceil log_2 I \rceil$), and controls the write and read instructions using the *write* and *word-enable* ports. The host selects the associative-search instruction using the twelve-element *control-in* port. It identifies the bit-slice(s) involved in the associative-search instruction using the J-element *slice-address-in* port (where $J=\lceil log_2 N \rceil$)and controls the *data-slice(s)* reading using the *slice-enable* port. The host retrieves the address of words satisfying the associative-search instruction using the K-element *address-out* port.

Within this chapter, each circuit description identifies potential critical-path delays for simulated word lengths of N equal to 12, 20, and 36 cells and I equal to 4, 8, 16, 32, 64, and 128 words. Insufficient computer memory prohibited the simulation of the entire architecture for every memory-array size. The simulations required two models. The first model resembled the SRAM simulation model. The bit-serial array is reduced to a single bit-slice of I-1 cells and a single N-cell word. This model was used to simulate the write, read, and bit-slice read operations. The second model simulated the processing elements using static loading to simulate the control delays.

5.3.1 Data-Write Circuit Description. The data-write circuit accepts the *data-in* and *write* port stimuli from the host to drive the *data* and $\overline{data}$ ports. It also accepts the slice address-decode circuit *bit-slice-select* port stimulus to form the *C-vect* port stimulus, which drives the PE array. The *bit-slice-select*, *data-in*, *data*, and $\overline{data}$ ports consist of N elements, and an element within each port is identified as *bit-slice-select(n)*, *data-in(n)*, *data(n)*, and $\overline{data(n)}$, respectively.

The data-write circuit is made of N write circuits. Figure 22 shows the write-circuit schematic for *data-in(n)*, and Table 31 gives its layout dimensions. Table 31 also gives the layout dimension of other circuits discussed within this section. The write-circuit schematic resembles the SRAM write-circuit schematic with additional circuitry to read the comparand bit into the PE array. The transistor-gate dimensions are the same as the SRAM write circuit, with the addition of the n-type transistor enabled by *bit-slice-select(n)* which has a 2 λ gate length and a 4 λ gate width.

The write and read operations are executed in a similar manner as explained in the SRAM architecture description. During the associative-search instruction, when *bit-slice-select(n)* is enabled, *data(n)* is passed to *C-vector(n mod bit-vector length)*; otherwise, *C-vector* is not driven. The bit-vector length is 1, 2, or 4 corresponding to the single, two, and

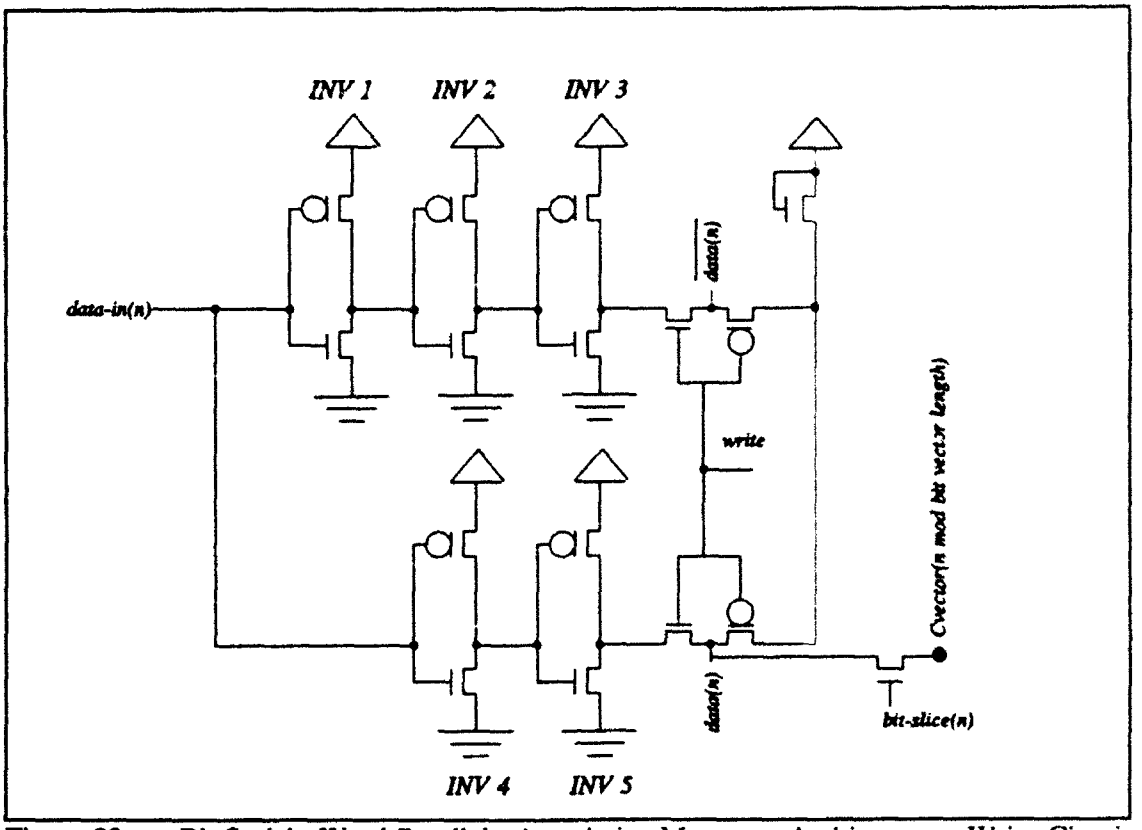

Figure 22. Bit-Serial Word-Parallel Associative-Memory Architecture Write-Circuit Schematic.

four BSWPAM architectures, respectively.

The time to form *data(n)* and $\overline{data(n)}$ has the potential to be a critical-path delay. The data, $\overline{data}$, and data-precharge delays are a function of the number of words in the bit-serial array. Tables 32, 33, and 34 give the delay information for the single, two, and four BSWPAM architecture write circuits, respectively. The single BSWPAM architecture *data(n)* and $\overline{data(n)}$ port fall delays require less time than the two BSWPAM architecture *data(n)* and $\overline{data(n)}$ ports, which in-turn require less time than the four BSWPAM architecture *data(n)* and $\overline{data(n)}$ ports. The fall-delay differences are attributed to the loads associated with each bit-serial array cell. As measured from the extracted SPICE netlist, the *data(n)* and $\overline{data(n)}$ ports associated with the single BSWPAM architecture have a capacitance of 11 fF/word. The *data(n)* and $\overline{data(n)}$ ports associated with the two BSWPAM architecture

Table 31

Bit-Serial Word-Parallel Associative-Memory
Component-Layout Dimensions.

Component	Height (λ)	Width (λ)
Four Bit-Serial Address Encoder	276	175
Four Bit-Serial Bit	99	40
Four Bit-Serial Decode Circuit	99	215
Four Bit-Serial Processing Element	276	537
Four Bit-Serial Read Circuit	50	40
Four Bit-Serial Slice-Decode Circuit	142	40
Four Bit-Serial Slice-Precharge Circuit	99	56
Four Bit-Serial Slice-Read Circuit	99	136
Four Bit-Serial Write Circuit	238	40
Driver (Address)	150	32
Single Bit-Serial Address Encoder	112	177
Single Bit-Serial Bit	60	40
Single Bit-Serial Decode Circuit	60	215
Single Bit-Serial Processing Element	112	601
Single Bit-Serial Read Circuit	50	40
Single Bit-Serial Slice-Decode Circuit	142	40
Single Bit-Serial Slice-Precharge Circuit	60	22
Single Bit-Serial Slice-Read Circuit	60	37
Single Bit-Serial Write Circuit	217	40
Two Bit-Serial Address Encoder	119	176
Two Bit-Serial Bit	74	40
Two Bit-Serial Decode Circuit	74	215
Two Bit-Serial Processing Element	119	769
Two Bit-Serial Read Circuit	50	40
Two Bit-Serial Slice-Decode Circuit	142	40
Two Bit-Serial Slice-Precharge Circuit	74	39
Two Bit-Serial Slice-Read Circuit	74	70
Two Bit-Serial Write Circuit	225	40

have a capacitance of 12 fF/word, and the *data(n)* and $\overline{data(n)}$ ports associated with the four BSWPAM architecture have a capacitance of 14 fF/word. The precharge delays show similar results. The single and two BSWPAM architecture precharge delays are approximately equal, while the four BSWPAM precharge delays are longer.

Table 32

Data, $\overline{Data}$, and Data Precharge Delays for the Single Bit-Serial Word-Parallel Associative-Memory Architecture.

Single Bit-Serial Write Circuit	data delay (ns) reference: *write* 50% pt	$\overline{data}$ delay (ns) reference: *write* 50% pt	precharge delay (ns) reference: *write* 50% pt
4 words	0 - 3.95 V $t_r < 0.5$ $t_f = 0.2$	0 - 3.95 V $t_r < 0.5$ $t_f = 0.2$	0 - 3.61 V $t_r = 5.1$
8 words	0 - 3.87 V $t_r < 0.5$ $t_f = 0.4$	0 - 3.87 V $t_r < 0.5$ $t_f = 0.4$	0 - 3.58 V $t_r = 7.6$
16 words	0 - 3.8 V $t_r < 0.5$ $t_f = 0.6$	0 - 3.8 V $t_r < 0.5$ $t_f = 0.6$	0 - 3.54 V $t_r = 12.7$
32 words	0 - 3.76 V $t_r < 0.6$ $t_f = 1.0$	0 - 3.76 V $t_r < 0.6$ $t_f = 1.0$	0 - 3.44 $t_r = 22.2$
64 words	0 - 3.73 V $t_r < 0.5$ $t_f = 1.8$	0 - 3.73 V $t_r < 0.6$ $t_f = 2.0$	0 - 3.21 V $t_r = 39.8$
128 words	0 - 3.71 V $t_r < 0.5$ $t_f = 2.9$	0 - 3.71 V $t_r < 0.6$ $t_f = 3.4$	0 - 2.7 V $t_r = 59.9$

Table 35 provides the least-square analysis of the delays. For each of the three data-write circuits, the data and $\overline{data}$ fall and precharge delays linearly increase as the number of words in each bit-serial array linearly increases. Table 35 also provides the least-square analysis for the other critical-path delays presented in this section.

5.3.2 Address-Driver Description. Each BSWPAM architecture uses the address driver presented in the SRAM architecture description to execute two functions. First, the address

Table 33

Data, $\overline{\text{Data}}$, and Data Precharge Delays for the
Two Bit-Serial Word-Parallel Associative-Memory Architecture.

Two Bit-Serial Write Circuit	data delay (ns) reference: *write* 50% pt range: 0 to 4.2 V	$\overline{\text{data}}$ delay (ns) reference: *write* 50% pt range: 0 to 4.2 V	precharge delay (ns) reference: *write* 50% pt range: 0 to 4.2 V
4 words	0 - 3.95 V $t_r < 0.5$ $t_f = 0.2$	0 - 3.95 V $t_r < 0.5$ $t_f = 0.2$	0 - 3.61 V $t_r = 4.8$
8 words	0 - 3.84 V $t_r < 0.5$ $t_f = 0.4$	0 - 3.84 V $t_r < 0.5$ $t_f = 0.4$	0 - 3.58 V $t_r = 8.1$
16 words	0 - 3.8 V $t_r < 0.5$ $t_f = 0.6$	0 - 3.8 V $t_r < 0.5$ $t_f = 0.6$	0 - 3.54 V $t_r = 12.9$
32 words	0 - 3.76 V $t_r < 0.6$ $t_f = 1.0$	0 - 3.76 V $t_r < 0.6$ $t_f = 1.0$	0 - 3.44 V $t_r = 22.9$
64 words	0 - 3.73 V $t_r < 0.5$ $t_f = 1.8$	0 - 3.73 V $t_r < 0.6$ $t_f = 1.9$	0 - 3.21 V $t_r = 41.0$
128 words	0 - 3.71 V $t_r < 0.5$ $t_f = 3.2$	0 - 3.71 V $t_r < 0.6$ $t_f = 3.7$	0 - 2.6 V $t_r = 61.8$

driver accepts the *address-in* port stimulus to form the *address* and $\overline{address}$ port stimuli. Second, the address driver accepts the *slice-address-in* port stimulus to form the *slice-address* and *slice-*$\overline{address}$ port stimuli.

Tables 36, 37, and 38 give the address and $\overline{\text{address}}$ delays for the single, two, and four BSWPAM architectures, respectively. The delays were collected for each architecture, but as expected, were nearly equal. This observation serves as a reasonability check to ensure that the measured delays are consistent between architectures of similar design. For each application of the address driver, the address and $\overline{\text{address}}$ rise and fall delays linearly increase as the number of words in the bit-serial array linearly increases. To show this linear dependency, Table 35 shows that the rise and fall-delay slopes for each architecture are equal,

Table 34

Data, $\overline{Data}$, and Data Precharge Delays for the
Four Bit-Serial Word-Parallel Associative-Memory Architecture.

Four Bit-Serial Write Circuit	data delay (ns) reference: *write* 50% pt	$\overline{data}$ delay (ns) reference: *write* 50% pt	precharge delay (ns) reference: *write* 50% pt
4 words	0 - 3.95 V $t_r < 0.5$ $t_f = 0.3$	0 - 3.95 V $t_r < 0.5$ $t_f = 0.3$	0 - 3.61 V $t_r = 6.1$
8 words	0 - 3.87 V $t_r < 0.5$ $t_f = 0.4$	0 - 3.87 V $t_r < 0.5$ $t_f = 0.4$	0 - 3.58 V $t_r = 9.6$
16 words	0 - 3.8 V $t_r < 0.5$ $t_f = 0.7$	0 - 3.8 V $t_r < 0.5$ $t_f = 0.7$	0 - 3.54 V $t_r = 16.3$
32 words	0 - 3.76 V $t_r < 0.6$ $t_f = 1.2$	0 - 3.76 V $t_r < 0.6$ $t_f = 1.3$	0 - 3.44 V $t_r = 29.1$
64 words	0 - 3.73 V $t_r < 0.5$ $t_f = 2.2$	0 - 3.73 V $t_r < 0.6$ $t_f = 2.4$	0 - 3.21 V $t_r = 48.7$
128 words	0 - 3.71 V $t_r < 0.5$ $t_f = 4.1$	0 - 3.71 V $t_r < 0.6$ $t_f = 4.7$	0 - 2.26 V $t_r = 65.4$

though the offsets differ by a few hundredths of a nanosecond. The offset difference between the rise and fall delay is attributed to the difference in the number of inverters necessary to form *address(k)* and $\overline{address(k)}$.

Table 39 gives the slice-address and slice-$\overline{address}$ delays for each architecture. The table shows that the delays are nearly constant between the three architectures. The slight increase in delays progressing from the single to the four BSWPAM architecture is attributed to the increase in capacitance associated with longer metal runs.

5.3.3 Address-Decode Circuit Description. Each BSWPAM architecture uses the address-decode circuit presented in Chapter III to execute two functions. First, it accepts the *address* and $\overline{address}$ port stimuli provided by the address driver to form the *I*-element port, *word-select*, used to select a bit-serial array word. Second, it accepts the *slice-address* and

Table 35

Least-Square Analysis for each
Bit-Serial Word-Parallel Associative-Memory Architecture.

Delays * 12\20\36-bit words ** 1\2\4 BSWPAM	Coefficient of Determination	Offset (ns)	Slope (ns/word)
Data Fall **	1.00\1.00\1.00	0.18\0.20\ 0.19	0.02\0.03\0.03
$\overline{\text{Data}}$ Fall **	1.00\1.00\1.00	0.13\0.22\ 0.14	0.03\0.03\0.04
Address Rise **	0.99\0.99\1.00	0.25\0.25\ 0.22	0.03\0.03\0.03
Address Fall **	0.99\0.99\0.99	0.60\0.57\0.57	0.01\0.01\0.01
$\overline{\text{Address}}$ Rise **	0.98\0.99\0.99	1.18\1.19\1.20	0.03\0.03\0.03
$\overline{\text{Address}}$ Fall **	1.00\1.00\0.99	0.97\1.00\0.97	0.01\0.01\0.01
Bit-Slice Select Rise **	1.00\1.00\1.00	1.94\1.93\1.90	0.06\0.07\0.10
Bit-Slice Select Fall **	1.00\1.00\0.99	2.08\2.07\2.24	0.03\0.04\0.04
Read Rise *	0.98\0.99\0.99	3.33\3.44\4.17	0.06\0.07\0.07
Read Fall *	0.99\0.99\0.96	6.89\7.09\7.66	0.52\0.52\0.52
1BSWPAM Slice Read Rise *	1.00\1.00\1.00	3.44\3.68\4.96	0.06\0.06\0.07
1BSWPAM Slice Read Fall *	1.00\0.99\0.96	7.75\10.74\ 18.60	0.04\0.04\0.05
2BSWPAM Slice Read Rise *	1.00\1.00\1.00	3.21\3.61\4.51	0.06\0.07\0.07
2BSWPAM Slice Read Fall *	1.00\1.00\1.00	7.22\10.23\ 16.34	0.05\0.05\0.04
4BSWPAM Slice Read Rise *	0.99\1.00\1.00	3.66\3.77\4.58	0.08\0.08\0.09
4BSWPAM Slice Read Fall *	1.00\1.00\1.00	7.44\8.69\13.58	0.07\0.07\0.07
Write *	0.95\0.99\0.97	6.28\6.75\7.53	0.03\0.03\0.03
Read *	1.00\0.99\0.99	6.07\6.62\7.65	0.09\0.09\0.09
1BSWPAM Slice Search *	1.00\1.00\1.00	9.54\9.78\12.06	0.06\0.06\0.07
2BSWPAM Slice Search *	1.00\1.00\1.00	9.20\10.01\ 11.81	0.06\0.07\0.07
4BSWPAM Slice Search *	0.99\1.00\1.00	9.76\10.17\ 11.88	0.08\0.08\0.09

slice-$\overline{address}$ port stimuli provided by the slice-address driver to form the *N*-element port, *bit-slice-select*, used to select a bit-slice of the bit-serial array.

Table 36

Address, $\overline{Address}$, and Address Decode Delays for the Single Bit-Serial Word-Parallel Associative-Memory Architecture.

Address Drive	address delay (ns) reference: *address-in* 50% pt range: 0 to 5 V	$\overline{address}$ delay (ns) reference: *address-in* 50% pt range: 0 to 5 V	decode delay (ns) reference: *address-in* 50% pt range: 1.3 to 0.1 V
4 words	t_r= 0.5 t_f= 0.6	t_r= 1.4 t_f= 1.0	t_f = 3.1
8 words	t_r= 0.6 t_f= 0.7	t_r= 1.5 t_f= 1.1	t_f = 3.3
16 words	t_r= 0.7 t_f= 0.7	t_r= 1.6 t_f= 1.2	t_f= 3.8
32 words	t_r= 1.1 t_f= 0.8	t_r= 1.9 t_f= 1.3	t_f = 4.5
64 words	t_r= 2.2 t_f= 1.1	t_r= 2.6 t_f= 1.7	t_f = 6.2
128 words	t_r= 4.4 t_f= 1.5	t_r= 4.7 t_f= 2.5	t_f = 8.8

The address-decode circuit used by each architecture is described in Chapter III, though the layout dimensions are altered to interface with each bit-serial array. Table 40 gives the word-select delays for each architecture. The word-select delays linearly increase with a linear increase in the number of bit-serial cells in a word.

The operation of the slice address-decode circuit differs between each architecture. For the single BSWPAM architecture, the value placed upon the *slice-address-in* port is used to select a single bit-slice of the bit-serial array. For example, if the host places 000011 onto *slice-address-in*, when the *slice-enable* port is enabled, *bit-slice-select(3)* will also be enabled.

For the two BSWPAM architecture, the value placed upon the *slice-address-in* port is used to select two bit-slices of the two bit-serial array. For example, if the host places

Table 37

Address, $\overline{\text{Address}}$, and Address Decode Delays for the Two Bit-Serial Word-Parallel Associative-Memory Architecture.

Address Driver	address delay (ns) reference: *address-in* 50% pt range: 0 to 5 V	$\overline{\text{address}}$ delay (ns) reference: *address-in* 50% pt range: 0 to 5 V	decode delay (ns) reference: *address-in* 50% pt range 1.3 to 0.1 V
4 words	t_r= 0.5 t_f= 0.6	t_r= 1.4 t_f= 1.0	t_f = 3.1
8 words	t_r= 0.6 t_f= 0.6	t_r= 1.5 t_f= 1.1	t_f = 3.3
16 words	t_r= 0.7 t_f= 0.7	t_r= 1.6 t_f= 1.2	t_f= 3.7
32 words	t_r= 1.1 t_f= 0.8	t_r= 1.9 t_f= 1.4	t_f = 4.5
64 words	t_r= 2.2 t_f= 1.1	t_r= 2.7 t_f= 1.8	t_f = 6.2
128 words	t_r= 4.4 t_f= 1.5	t_r= 4.7 t_f= 2.5	t_f = 8.9

000011 onto *slice-address-in*, when the *slice-enable* port is enabled, *bit-slice-select(6)* and *(7)* will be enabled. In general, the value the host places onto *slice-address-in*, A, will enable *bit-slice-select(2A)* and *(2A+1)*.

For the four BSWPAM architecture, the value placed upon the *address-in* port is used to select four bit-slices of the four bit-serial array. For example, if the host places 000011 onto *slice-address-in*, when the *slice-enable* port is enabled, *bit-slice-select(12)*, *(13)*, *(14)*, and *(15)* will be enabled. In general, the value the host places onto *slice-address-in*, *A*, will enable *bit-slice-select(4A)*, *(4A+1)*, *(4A+2)*, *(4A+3)*.

The bit-slice delays are referenced from the *slice-enable* port. The bit-slice select rise delay is the propagation delay necessary for the *bit-slice-select(n)* port to reach the 50% point in its transition from ground to V_{dd}. The bit-slice select fall delay is the propagation delay necessary for the *bit-slice-select(n)* port to reach the 50% point in its transition from V_{dd} to ground. Table 41 gives the bit-slice select delays for each architecture. The bit-slice delays

Table 38

Address, $\overline{\text{Address}}$, and Address Decode Delays for the Four Bit-Serial Word-Parallel Associative-Memory Architecture.

Address Driver	address delay (ns) reference: *address-in* 50% pt range: 0 to 5 V	$\overline{\text{address}}$ delay (ns) reference: *address-in* 50% pt range: 0 to 5 V	decode delay (ns) reference: *address-in* 50% pt range: 1.3 to 0.1 V
4 words	t_r=0.5 t_f=0.6	t_r=1.4 t_f=1.0	t_f = 3.1
8 words	t_r=0.5 t_f=0.6	t_r=1.5 t_f=1.0	t_f = 3.3
16 words	t_r=0.7 t_f=0.7	t_r=1.6 t_f=1.2	t_f= 3.7
32 words	t_r=1.1 t_f=0.8	t_r=1.9 t_f=1.4	t_f = 4.5
64 words	t_r=2.2 t_f=1.1	t_r=2.7 t_f=1.8	t_f = 6.2
128 words	t_r=4.4 t_f=1.5	t_r=4.6 t_f=2.5	t_f = 8.8

linearly increase as the number of words linearly increases. Also, the delays increase from the single to the four BSWPAM architecture.

Table 35 shows a nearly constant offset, and an increasing slope progressing from the single to the four BSWPAM architecture. The increasing slope is attributed to the increasing load associated with driving a single bit slice of the single bit-serial array to driving four bit-slices of the four bit-serial array.

5.3.4 Bit-Serial Array Description. Each BSWPAM architecture has an individually designed bit-serial array. Figure 23 shows the bit-serial array associated with the single BSWPAM architecture. It contains *I word-select, data-slice*, and $\overline{data}$*-slice* lines, identified as *word-select(0)* to *word-select(I-1)*, *data-slice(0)* to *data-slice(I-1)* and $\overline{data}$*-slice(0)* to $\overline{data}$*-slice(I-1)*. It also contains *N bit-slice-select, data*, and $\overline{data}$ lines using the same nomenclature as presented in the data-write circuit description. A bit-serial cell is identified as *cell(i, n)*.

Table 39

Slice Address, Slice $\overline{\text{Address}}$, and Slice Address Decode Delays for each BSWPAM Architecture.

Slice Driver	slice address delay (ns) reference: *slice-address-in* 50% pt range: 0 to 5 V	slice $\overline{\text{address}}$ delay (ns) reference: *slice-address-in* 50% pt range: 0 to 5 V	slice decode delay (ns) reference: *slice-address-in* 50% pt range:1.3 to 0.1 V
Single Bit-Serial byte	t_r=0.4 t_f=0.5	t_r=1.9 t_f=1.4	t_f = 3.7
Single Bit-Serial half-word	t_r=0.4 t_f=0.5	t_r=1.9 t_f=1.5	t_f = 3.8
Single Bit-Serial word	t_r=0.5 t_f=0.6	t_r=2.9 t_f=1.9	t_f = 4.4
Two Bit-Serial byte	t_r=0.4 t_f=0.5	t_r=1.8 t_f=1.3	t_f = 3.5
Two Bit-Serial half-word	t_r=0.4 t_f=0.5	t_r=2.2 t_f=1.5	t_f = 3.8
Two Bit-Serial word	t_r=0.4 t_f=0.5	t_r=3.1 t_f=2.0	t_f = 4.6
Four Bit-Serial byte	t_r=0.4 t_f=0.6	t_r=1.9 t_f=1.4	t_f = 3.8
Four Bit-Serial half-word	t_r=0.4 t_f=0.5	t_r=2.2 t_f=1.5	t_f = 4.1
Four Bit-Serial word	t_r=0.6 t_f=0.6	t_r=3.1 t_f=1.9	t_f = 4.8

Figure 24 shows the bit-serial array associated with the two BSWPAM architecture. It operates in the same manner as the bit-serial array associated with the single BSWPAM architecture except in the way it drives the *data-slice* ports. When *bit-slice-select(n)* is enabled, the content of *cell(i, n)* is passed to *data-slice(n mod 2)(i)* and the inverse of the content of *cell(i, n)* is passed to $\overline{\textit{data}}$*-slice(n mod 2)(i)*.

Table 40

Word-Select Delays for each
Bit-Serial Word-Parallel Associative-Memory Architecture.

Decode Circuit	word-select delay (ns) 12 bits reference: *word-enable* 50% pt range: 0 to 5 V	word-select delay (ns) 20 bits reference: *word-enable* 50% pt range: 0 to 5 V	word-select delay (ns) 36 bits reference: *word-enable* 50% pt range: 0 to 5 V
4 words	t_r=1.8 t_f=1.9	t_r=2.0 t_f=2.0	t_r=2.5 t_f=2.3
8 words	t_r=1.8 t_f=1.9	t_r=2.0 t_f=2.0	t_r=2.5 t_f=2.3
16 words	t_r=1.8 t_f=1.9	t_r=2.0 t_f=2.0	t_r=2.5 t_f=2.3
32 words	t_r=1.8 t_f=1.8	t_r=2.0 t_f=2.0	t_r=2.6 t_f=2.3
64 words	t_r=1.8 t_f=1.8	t_r=2.0 t_f=2.0	t_r=2.6 t_f=2.3
128 words	t_r=2.0 t_f=1.8	t_r=2.1 t_f=2.0	t_r=2.6 t_f=2.3

Figure 25 shows the bit-serial array associated with the four BSWPAM architecture. It operates in the same manner as the bit-serial array associated with the single BSWPAM architecture, except when *bit-slice-select(n)* is enabled, the content of *cell(i, n)* is passed to *data-slice(n mod 4)(i)*, and the inverse of the content of *cell(i, n)* is passed to $\overline{data}$*-slice(n mod 4)(i)*.

Figure 26 shows the bit-serial cell. Each transistor in the cell is designed with a 2 λ channel length and a 3 λ channel width. The cell operation is similar to the SRAM cell operation except for the additional capability to read the cell content to the *data-slice* and $\overline{data}$*-slice* ports. The cell layout dimensions affect the dimensions of the remaining circuits. The cell width constrains the width of the write, read, and slice-decode circuits. The cell height constrains the height of the decode, and slice-read circuits.

Table 41

Bit-Slice Select Delays for each
Bit-Serial Word-Parallel Associative-Memory Architecture.

Decode Circuit	single bit-serial bit-slice select delay (ns) reference: *slice-enable* 50% pt range: 0 to 5 V	two bit-serial bit-slice select delay (ns) reference: *slice-enable* 50% pt range: 0 to 5 V	four bit-serial bit-slice select delay (ns) reference: *slice-enable* 50% pt range: 0 to 5 V
4 words	t_r= 2.2 t_f= 2.2	t_r= 2.2 t_f= 2.2	t_r= 2.3 t_f= 2.2
8 words	t_r= 2.4 t_f= 2.3	t_r= 2.5 t_f= 2.4	t_r= 2.7 t_f= 2.5
16 words	t_r= 2.9 t_f= 2.6	t_r= 3.0 t_f= 2.6	t_r= 3.4 t_f= 2.9
32 words	t_r= 3.8 t_f= 3.0	t_r= 4.0 t_f= 3.2	t_r= 4.9 t_f= 3.7
64 words	t_r= 5.7 t_f= 4.0	t_r= 6.3 t_f= 4.3	t_r= 8.0 t_f= 5.3
128 words	t_r= 9.5 t_f= 5.9	t_r=10.5 t_f= 6.6	t_r=14.1 t_f= 7.4

5.3.5 Data-Read Circuit and Slice Data-Read Circuit Description. Each BSWPAM architecture uses the data-read circuit presented in Chapter III to execute two functions. First, it is used to accept the *data* and $\overline{data}$ port stimuli to form *data-out*. Second, the single BSWPAM architecture uses the data-read circuit to accept the *data-slice* and $\overline{data}$*-slice* port stimuli to form *data-slice-out*. The two BSWPAM architecture uses two data-read circuits. The first data-read circuit ports are *data-slice-0(i)*, $\overline{data}$*-slice-0(i)* and *data-slice-out-0(i)*, and the second data-read circuit ports are *data-slice-1(i)*, $\overline{data}$*-slice-1(i)*, and *data-slice-out-1(i)*. Likewise, the four BSWPAM architecture uses four data-read circuits using the same port map scheme as discussed for the two BSWPAM architecture.

Chapter III defines the read delays, and Table 42 gives the values. As is seen from the table, the read delays linearly increase as the number of cells in a word and the number of words in the bit-serial array linearly increase.

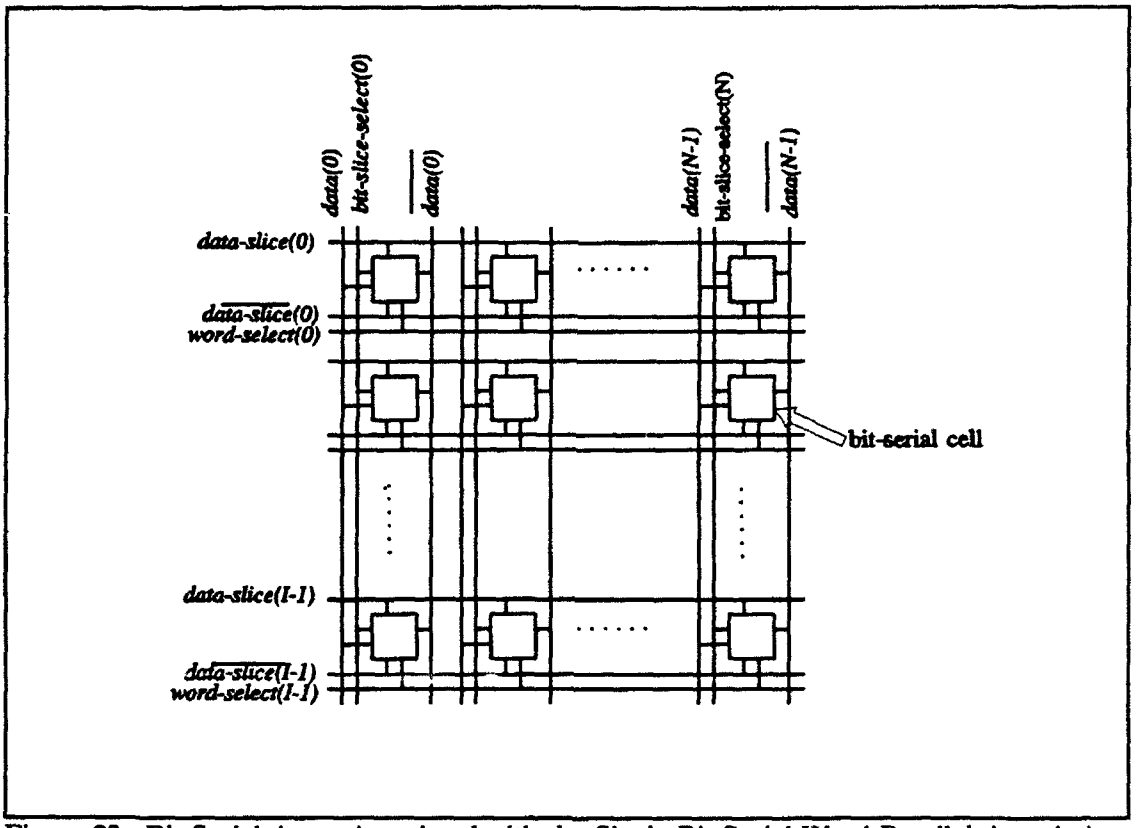

Figure 23. Bit-Serial Array Associated with the Single Bit-Serial Word-Parallel Associative-Memory Architecture.

The slice-read delays are referenced from the *slice-enable* port. The slice-read rise delay is the propagation delay necessary for the *data-slice-out* port to reach the 50% point in its transition from a steady-state voltage of approximately 1.6 V to a high value of approximately 4.2 V. The slice-read fall delay is the propagation delay necessary for the *data-slice-out* port to reach the 50% point in its transition from a high value to steady-state without external stimulus. The transition from steady-state to a low value using external stimulus was consistently less than 0.5 ns.

The slice-read delays linearly increase as the number of cells in a word and the number of words in the bit-serial array linearly increases. Tables 43, 44, and 45 give the slice-read delays for the single, two, and four BSWPAM architectures, respectively. The slice-read delays are nearly constant from the single to the four BSWPAM architecture. This can be

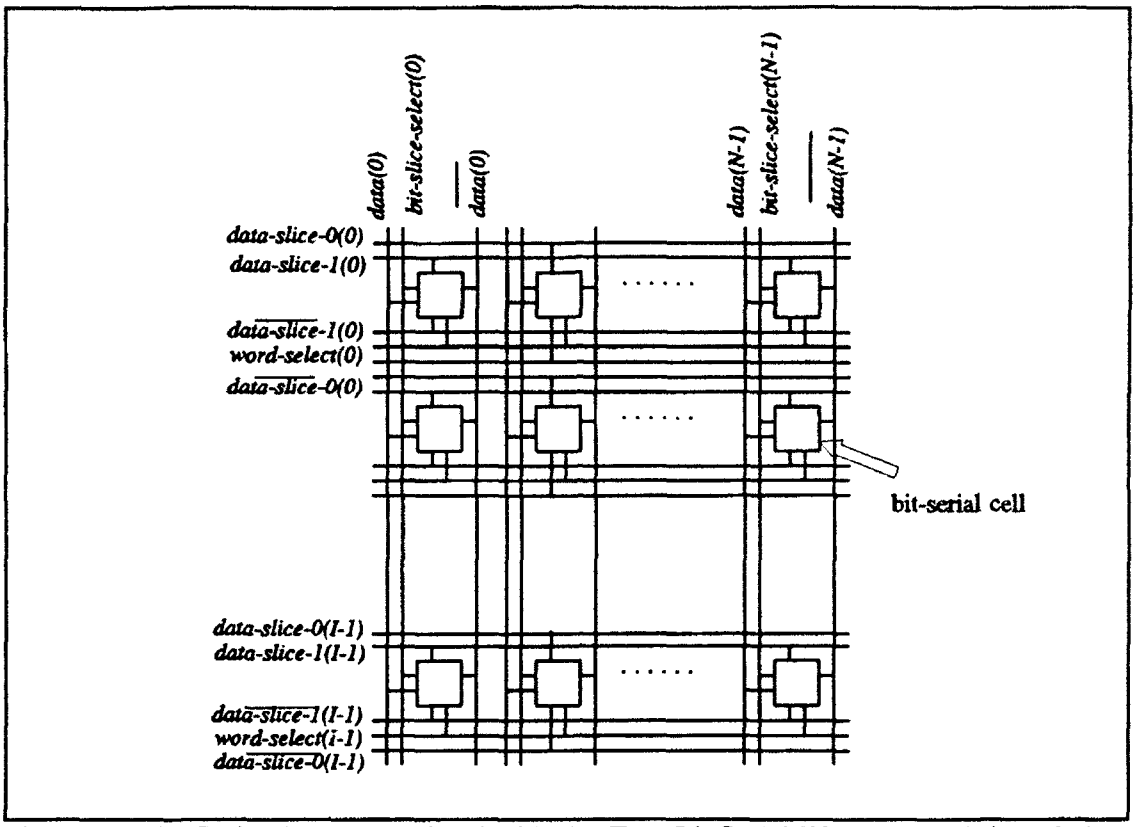

Figure 24. Bit-Serial Array Associated with the Two Bit-Serial Word-Parallel Associative-Memory Architecture.

explained by considering the loads necessary to read the bit-serial array. The *bit-slice-select* port load doubles from the single to the two BSWPAM architecture, and doubles again to the four BSWPAM architecture, so this increased load should increase the slice-read delays. On the other hand, the load on the *data-slice* and $\overline{data}$*-slice* ports is reduced by one-half from the single to the two BSWPAM architecture and is reduced by one-half again to the four BSWPAM architecture.

The data reveals that the reduced load on the *data-slice* and $\overline{data}$*-slice* ports of the two BSWPAM architecture as compared to the single BSWPAM architecture dominates and reduces the overall read time. Likewise, the increased load on the *bit-slice-select* port of the four BSWPAM architecture as compared to the single BSWPAM architecture increases the delay.

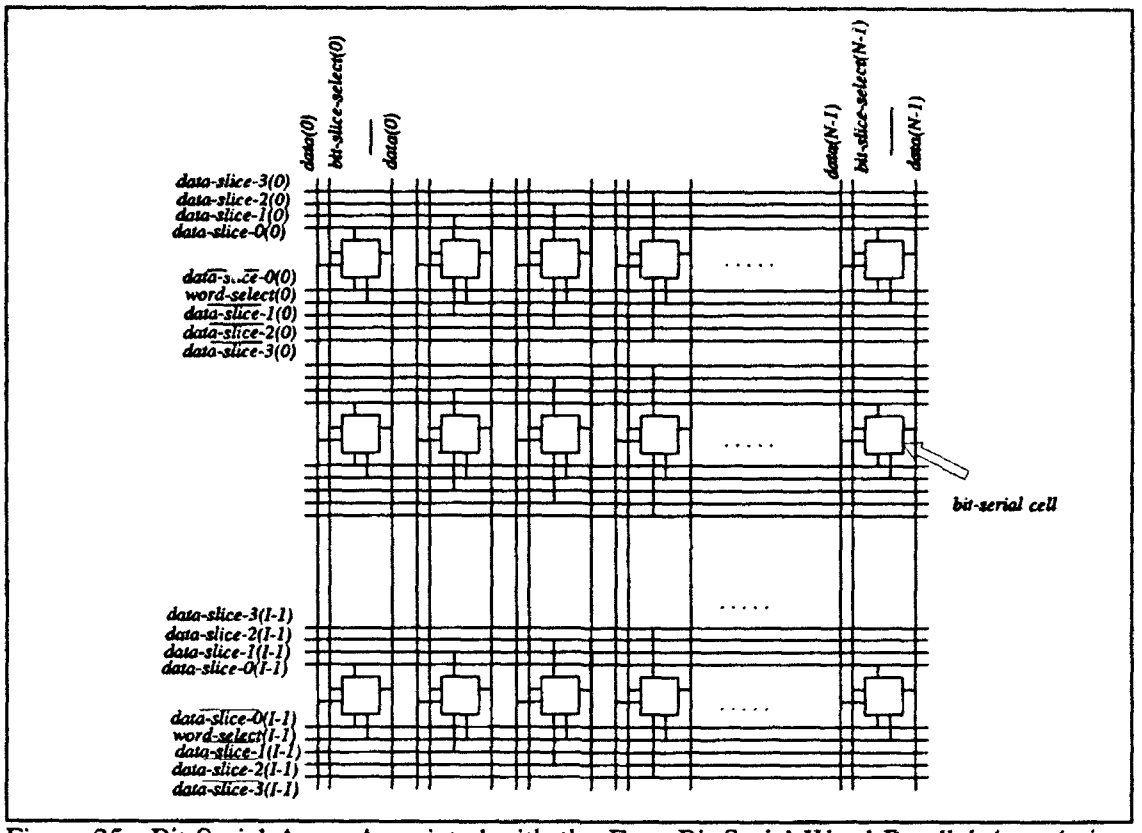

Figure 25. Bit-Serial Array Associated with the Four Bit-Serial Word-Parallel Associative-Memory Architecture.

5.3.6 PE-Array Description. Each BSWPAM architecture has an individually designed PE array to accommodate the different amounts of data processed. Each PE array bit-serially accepts elements from the comparand and from the slice data-read circuit(s) to execute associative-search operations on the bit-serial array contents.

For each PE array, the host drives the *control-in* port. The data-write circuit drives the PE-array *C1*, *C2*, *C3*, and *C4* ports. The slice data-read circuit drives the PE-array *D1*, *D2*, *D3*, and *D4* ports, and each PE array forms *Ri-out* and *C-out*. Each PE array consists of three subcircuits: the control-1 driver, the control-2 driver, and *I* processing elements beginning with *PE(0)* and concluding with *PE(I-1)*. The control-1 and control-2 drivers accept the *control-in* port stimulus to form the *control* and $\overline{control}$ port stimuli used by the processing elements. Each BSWPAM architecture uses the control-1 and control-2 drivers

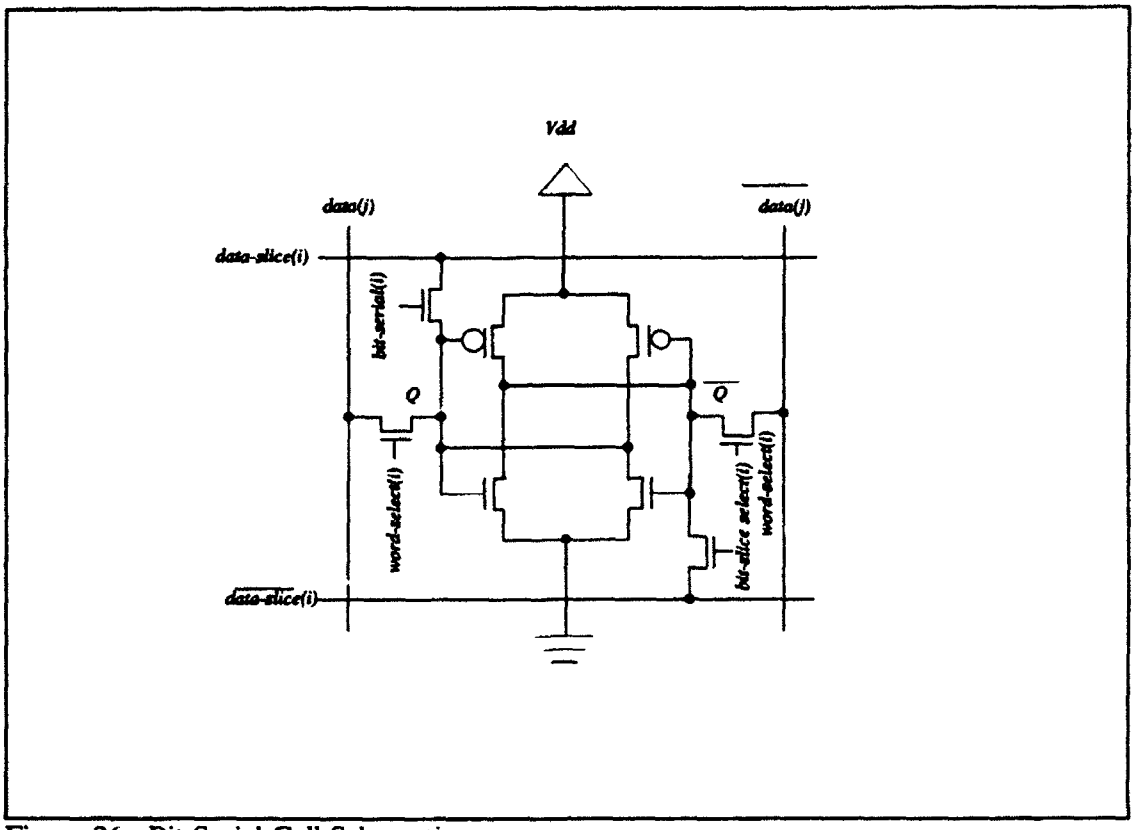

Figure 26. Bit-Serial Cell Schematic.

discussed in Chapter IV. *Control-in(0)*, *(1)*, *(2)*, *(4)*, *(5)*, *(7)*, *(8)*, *(9)*, *(10)*, and *(11)* are inputs to the control-1 driver forming *control(0)*, *(1)*, *(2)*, *(4)*, *(5)*, *(7)*, *(8)*, *(9)*, *(10)*, and *(11)*. *Control-in(2)*, and *(6)* are inputs to control-2 drivers forming *control(2)*, *control(6)*, $\overline{control}(2)$, and $\overline{control}(6)$.

Figure 27 shows the single BSWPAM PE schematic, Figure 28 shows the two BSWPAM PE schematic, and Figure 29 shows the four BSWPAM PE schematic. Table 46 defines the *control-in* port functions. Table 47 gives the delays required by each BSWPAM PE array to load, store, compare, and transfer data. The delays are measured for a single processing element and do not include the times required to form the *control* and $\overline{control}$ ports. The results are not comparable between architectures. Though each PE array is similar in concept, the layout used to form the extracted SPICE netlist are different. Each

Table 42

Read Delays for the
Bit-Serial Word-Parallel Associative-Memory Architectures.

Read Circuit	read delay (ns) 12 bits reference: *word-enable* 50% pt range: 1.2 to 4.1 V (unless otherwise stated)	read delay (ns) 20 bits reference: *word-enable* 50% pt range: 1.2 to 4.1 V (unless otherwise stated)	read delay (ns) 36 bits reference: *word-enable* 50% pt range: 1.2 to 4.1 V (unless otherwise stated)
4 words	t_r= 3.1 t_f= 7.1	t_r= 3.4 t_f= 7.3	t_r= 4.0 t_f= 7.7
8 words	t_r= 3.4 t_f= 9.6	t_r= 3.8 t_f= 9.9	t_r= 4.5 t_f=10.2
16 words	t_r= 4.2 t_f=14.8	t_r= 4.6 t_f=15.0	t_r= 5.3 t_f=15.3
32 words	t_r= 5.3 t_f=24.9	t_r= 5.8 t_f=25.2	t_r= 6.7 t_f=26.8
64 words	1.4 - 4.2 V t_r= 7.3 t_f= 44.2	1.4 - 4.1 V t_r= 7.9 t_f= 44.6	1.4 - 4.1 V t_r= 8.7 t_f= 45.0
128 words	2.4 - 4.2 V t_r=11.0 t_f= 70.8	1.8 - 4.1 V t_r=11.6 t_f= 70.9	1.8 - 4.1 V t_r=12.2 t_f= 71.4

PE array executes their operations in approximately the same time. This result is attributed to the nearly constant number of logic levels between the three BSWPAM processing elements. Though the number of gates per level increases from the single to the four BSWPAM PE, the number of logic levels is nearly constant.

5.3.7 Address-Encoder Description. Each BSWPAM architecture uses the same address encoder, though the layout dimensions are different to accommodate the array dimensions. Each architecture uses *Ri-out* and *C-out* from the PE array and *control(11)* from the control-1 driver to form the address of the first word in the bit-serial array that satisfactorily meets the search criteria. This address is made available to the host for further processing.

Table 43

Slice Read Delays for the
Single Bit-Serial Word-Parallel Associative-Memory Architecture.

Read Circuit	slice read delay (ns) 12 bits reference: *slice-enable* 50% pt range: 1.6 to 4.2 V	slice read delay (ns) 20 bits reference: *slice-enable* 50% pt range: 1.5 to 4.2 V	slice read delay (ns) 36 bits reference: *slice-enable* 50% pt range: 1.5 to 4.2 V
4 words	t_r= 3.5 t_f= 8.0	t_r= 3.8 t_f=10.8	t_r= 5.1 t_f=18.2
8 words	t_r= 3.8 t_f= 8.1	t_r= 4.1 t_f=10.8	t_r= 5.5 t_f=18.7
16 words	t_r= 4.3 t_f= 8.5	t_r= 4.6 t_f=11.6	t_r= 5.9 t_f=20.0
32 words	t_r= 5.4 t_f= 9.1	t_r= 5.7 t_f=12.3	t_r= 7.2 t_f=20.5
64 words	t_r= 7.2 t_f=10.5	t_r= 7.7 t_f=13.5	t_r= 9.4 t_f=21.6
128 words	t_r=10.4 t_f=13.5	t_r=11.2 t_f=16.0	t_r=13.3 t_f=24.3

The address encoder is a concatenation of *I* encoders, beginning with *encoder(0)* and concluding with *encoder(I-1)*. *Encoder(0)* uses *Ri-out(0)* from *PE(0)* and *control(11)* from the host to control the encoding process. All other encoders use *Ri-out(i)* from *PE(i)* and *c-out(i)* to control the encoding process. The address encoder forms the *address-out* port from this stimuli.

Figure 30 shows the encoder schematic for the 128th word of a 128-word bit-serial array. Table 48 gives the transistor-gate dimensions. The encoder consists of an AND-gate that drives an inverter which in turn is used to activate the seven t-gates. If both AND-gate inputs are enabled, then the t-gates either disable or enable the *address-out(k)* port; otherwise, the t-gates are disconnected and do not affect the *address-out* ports.

The address encoder gives an unambiguous address because only one encoder will have both AND-gate inputs enabled at a time. The PE array ensures this condition. While

Table 44

Slice Read Delays for the
Two Bit-Serial Word-Parallel Associative-Memory Architecture.

Read Circuit	slice read delay (ns) 12 bits reference: *slice-enable* 50% pt range: 1.5 to 4.2 V	slice read delay (ns) 20 bits reference: *slice-enable* 50% pt range: 1.5 to 4.2 V	slice read delay (ns) 36 bits reference: *slice-enable* 50% pt range: 1.5 to 4.2 V
4 words	t_r= _ t_f= 7.5	t_r= 3.7 t_f=10.5	t_r= 4.6 t_f=16.5
8 words	t_r= 3.6 t_f= 7.6	t_r= 4.0 t_f=10.6	t_r= 5.0 t_f=16.7
16 words	t_r= 4.2 t_f= 7.9	t_r= 4.7 t_f=10.9	t_r= 5.7 t_f=17.0
32 words	t_r= 5.3 t_f= 8.6	t_r= 5.9 t_f=11.7	t_r= 7.0 t_f=17.8
64 words	t_r= 7.2 t_f=10.1	t_r= 8.0 t_f=13.0	t_r= 9.4 t_f=18.9
128 words	t_r=10.6 t_f=13.1	t_r=11.8 t_f=16.1	t_r=13.6 t_f=21.8

the *control-in(11)* port is enabled, suppose more than one processing element matches an associative-search operation; therefore, *Ri-out* is enabled for more than one PE. The *control(11)* port output is passed serially from one processing element to another, starting with *PE(0)* to the first processing element with an enabled output. All processing elements following that first processing element with an enabled output receive a disabled *control(11)* stimulus. Therefore, all processing elements above the first matching processing element may receive an enabled *c-out* stimulus, but the *Ri-out* values are disabled, and all processing elements following the first matching processing element receive a disabled *c-out* stimulus. The time required to form *address-out* is shown in Table 49.

5.4 Bit-Serial Word-Parallel Associative-Memory Operation Description

Each BSWPAM architecture operates in one of two modes, word or phrase. They can execute one of eight instructions in each mode: write, read, $Op_=(C(s), db(s))$,

Table 45

Slice Read Delays for the
Four Bit-Serial Word-Parallel Associative-Memory Architecture.

Read Circuit	slice read delay (ns) 12 bits reference: *slice-enable* 50% pt range: 1.5 to 4.2 V	slice read delay (ns) 20 bits reference: *slice-enable* 50% pt range: 1.5 to 4.2 V	slice read delay (ns) 36 bits reference: *slice-enable* 50% pt range: 1.5 to 4.2 V
4 words	t_r= 3.7 t_f= 7.8	t_r= 3.8 t_f= 8.9	t_r= 4.6 t_f=13.9
8 words	t_r= 4.1 t_f= 8.1	t_r= 4.3 t_f= 9.3	t_r= 5.2 t_f=14.1
16 words	t_r= 5.0 t_f= 8.6	t_r= 5.2 t_f= 9.8	t_r= 6.2 t_f=14.6
32 words	t_r= 6.4 t_f= 9.7	t_r= 6.7 t_f=10.8	t_r= 7.9 t_f=15.6
64 words	t_r= 9.1 t_f=12.0	t_r= 9.5 t_f=13.0	t_r=11.0 t_f=12.8
128 words	t_r=13.5 t_f=16.9	t_r=14.3 t_f=17.3	t_r=16.5 t_f=22.0

$Op_{\neq}(C(s), db(s))$, $Op_{<}(C(s), db(s))$, $Op_{\leq}(C(s), db(s))$, $Op_{>}(C(s), db(s))$, and $Op_{\geq}(C(s), db(s))$. The process to execute each instruction is discussed next.

5.4.1 Word Mode. In the word mode, each BSWPAM architecture can perform a single write and read instruction, and a variety of associative-search instructions with each requiring three phases. First, the write and read instructions are presented. Then the associative-search instructions are presented.

5.4.1.1 Write and Read Instructions. The host can execute a single write and read instruction. Both instructions require three phases. The PE array is not involved in either instruction, so *control-in* is set to 000000000000 during the three phases. Likewise, the slice address circuitry is not involved, so *slice-enable* is disabled. For the write instruction, the first phase begins when the host places the data onto *data-in*, places the address onto *address-in*, and enables the *write* port. For the read instruction, the first phase begins when

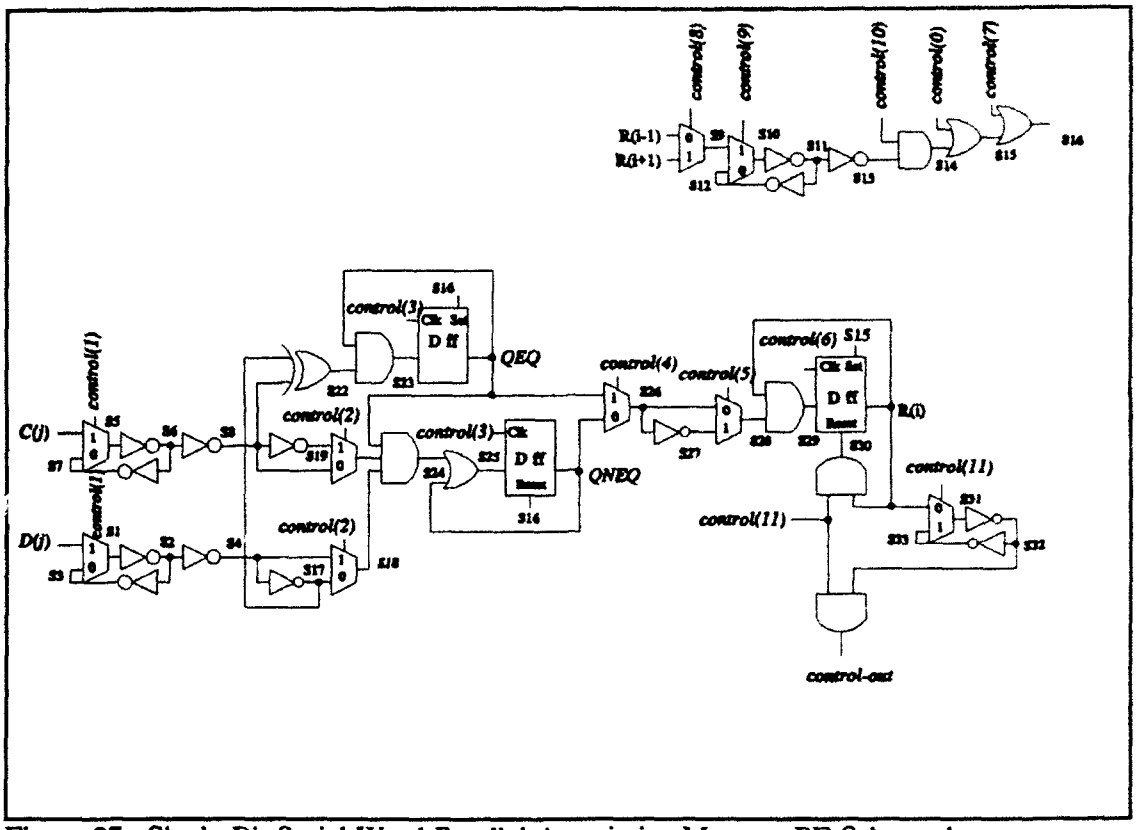

Figure 27. Single Bit-Serial Word-Parallel Associative-Memory PE Schematic.

the host places the address onto *address-in*. Once sufficient time has passed to form *data*, $\overline{data}$, *address*, and $\overline{address}$, the host begins the second phase by enabling the *word-enable* port. Once sufficient time has passed to write the data into or read data from the bit-serial array, the host begins the third phase by disabling the *word-enable* port, and for the write instruction, disabling the *write* port. After every element of the *word-select* port is disabled, the host can begin a new instruction.

Table 50 gives the delays to execute a write instruction. The write delay is calculated by summing the maximum value of the address, and $\overline{address}$ delays from Tables 36 through 38, the word-select rise delay from Table 40, the time to write data into the bit-serial array, and the word-select fall delay again from Table 40. For all three word lengths, the write delays linearly increase as the number of words in the bit-serial array linearly increases.

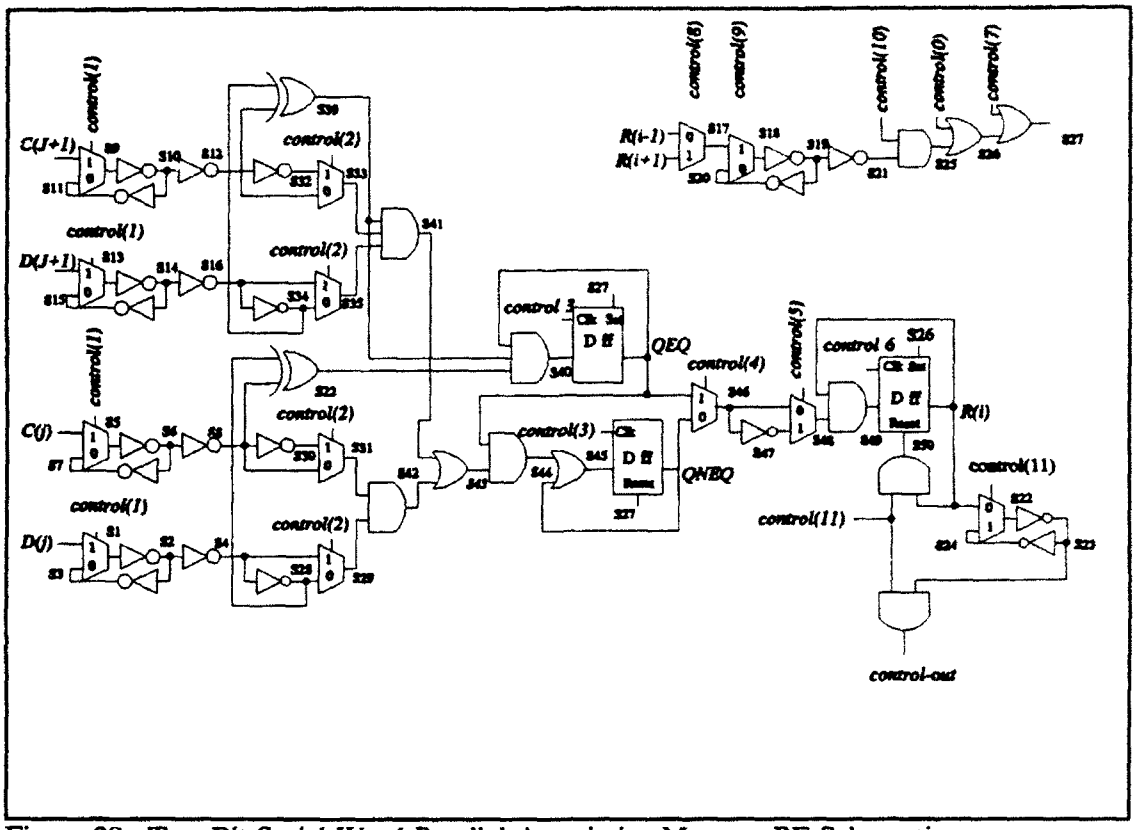

Figure 28. Two Bit-Serial Word-Parallel Associative-Memory PE Schematic.

Table 35 shows that the slopes are nearly constant with the offset linearly increasing with a linear increase in the number of bit-serial cells in a word.

The least-square analysis is performed for 12, 20, and 36-cell word lengths. From a least-square analysis of the three offsets, a single equation characterizing the write instruction is formed. Equation (51) approximates the BSWPAM write-instruction execution time.

$$\tau_{write}=(5.695+0.051(T(s)+T_{wdf}+T_{pp})+0.028l)\ ns \qquad (51)$$

As expected, the delay depends upon the number of cells in a word and the number of words in the bit-serial array. The percentage difference between the equation and the tabulated write times is measured to be less than 5.2% for all simulated bit-serial array sizes.

Table 51 gives the delays to execute a read instruction. The read delay is calculated by summing the maximum value of the address, and $\overline{\text{address}}$ delay from Tables 36 through

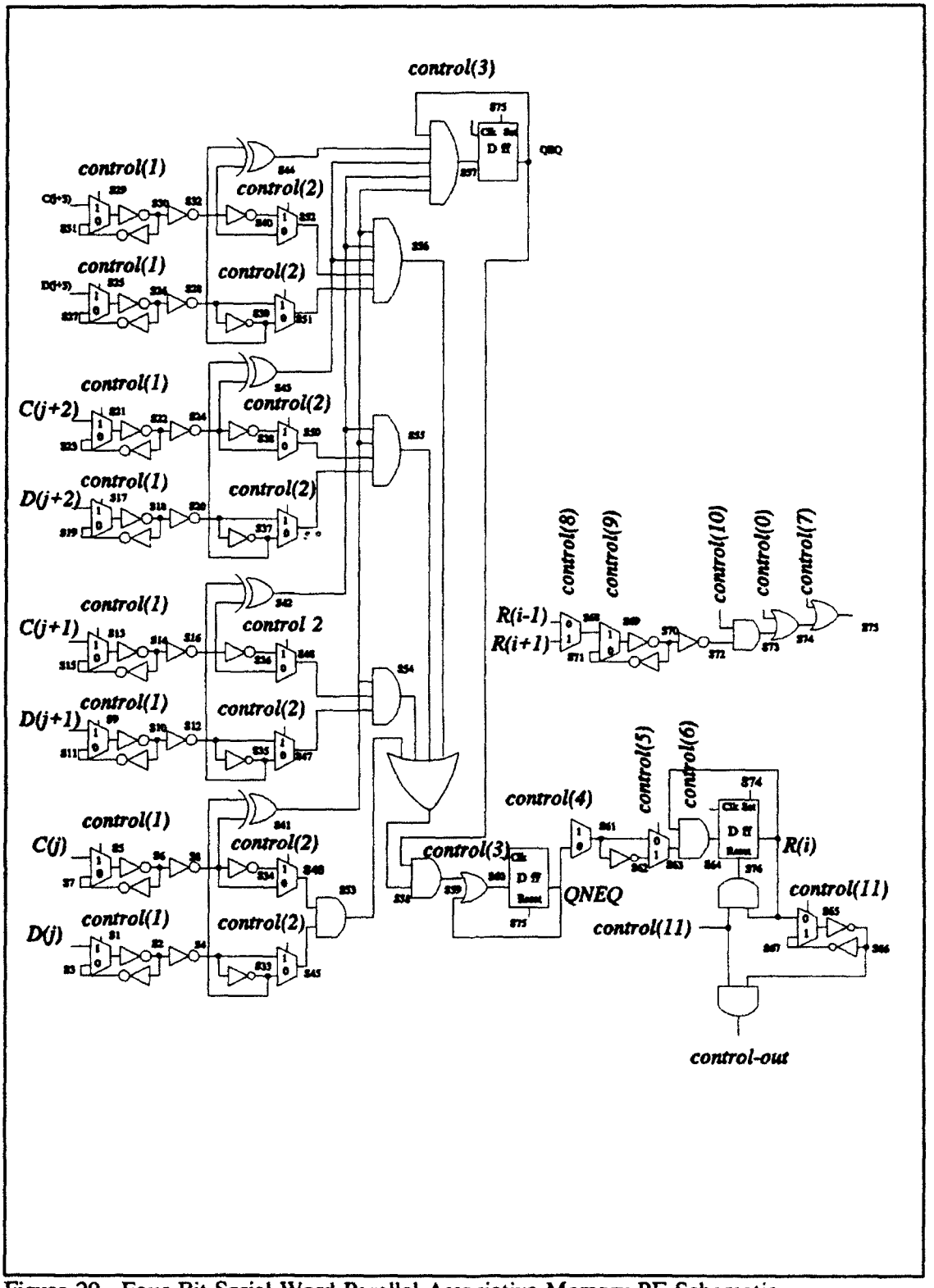

Figure 29. Four Bit-Serial Word-Parallel Associative-Memory PE Schematic.

Table 46

PE Array *Control* Port Functions.

Port	Function	Disabled	Enabled
control-in(0)	Global Initialization	No Op	Execute
control-in(1)	Load *C1* and *D1*	No Op	Load
control-in(2)	Select greater-than or less-than instruction	Less-than	Greater-than
control-in(3)	Store Eq, Ineq Registers	No Op	Store
control-in(4)	Select *QEQ*, *QNEQ*	*QNEQ*	*QEQ*
control-in(5)	Pass *control(4)* or inverse of *control(4)* selection	*control(4)*	inverse
control-in(6)	Store Word Search Register	No Op	Load
control-in(7)	Partial Initialization	No Op	Initialize
control-in(8)	Select *R(i-1)* or *R(i+1)*	*R(i-1)*	*R(i+1)*
control-in(9)	Store *control(7)* selection	No Op	Store
control-in(10)	Conditional Full Initialization	No Op	Initialize
control-in(11)	Word Search Register Conditional Reset	No Op	Reset

38, the read rise delay from Table 42, and the word-select fall delay from Table 40. For all three word lengths, the read delays linearly increase as the number of words in the bit-serial array linearly increases. Table 35 shows that the slopes are nearly constant and the offsets linearly increase with a linear increase in the number of cells in a word. Equation (52) approximates the BSWPAM read instruction execution time.

$$\tau_{read}=(5.29+0.066(T(s)+T_{wdf}+T_{pf})+0.091l)\ ns \qquad (52)$$

The percentage difference between the equation and the tabulated read times is measured to be less than 4.3% for all simulated bit-serial array sizes.

Table 47

Bit-Serial Word-Parallel Associative-Memory
Processing-Element Search Delays.

PE Timing Delays (ns)	Single Bit-Serial Word-Parallel AM	Two Bit-Serial Word-Parallel AM	Four Bit-Serial Word-Parallel AM
Data Load and Evaluation Ref: for t_r *control-in(1)* for t_f *control-in(7)*	measured at max(S23, S25) t_r=4.2 t_f=2.7	measured at max(S40, S45) t_r=5.3 t_f=3.6	measured at max(S57, S60) t_r=5.2 t_f=4.5
EQ/INEQ Reg Store and Xsfer Ref: for t_r *control-in(7)* for t_f *control-in(3)*	measured at S29 t_r=5.2 t_f=0.9	measured at S49 t_r=4.3 t_f=4.5	measured at S64 t_r=5.2 t_f=4.5
Word Search Store Ref: for t_r *control-in(10)* for t_f *control-in(6)*	t_r=5.2 t_f=1.3	t_r=4.3 t_f=1.4	t_r=5.4 t_f=1.5
Word Search Transfer Ref: *control-in(9)*	measured at S13 t_r=1.8 t_f=1.3	measured at S21 t_r=2.1 t_f=1.1	measured at S72 t_r=2.1 t_f=1.3
Time to form control-out Ref: *control-in(11)*	t_r=1.2 t_f=0.8	t_r=1.1 t_f=0.8	t_r=1.5 t_f=1.1

5.4.1.2 Associative-Search Instructions. The host can select one of six associative-search instructions, $OP_{=}(C(s), db(s))$, $Op_{\neq}(C(s), db(s))$, $Op_{<}(C(s), db(s))$, $Op_{>}(C(s), db(s))$, $Op_{\leq}(C(s), db(s))$, $Op_{\geq}(C(s), db(s))$. Each associative-search instruction for each BSWPAM architecture begins when the host places onto the *data-in* port the comparand, enables the *write* port, selects the valid-data field to compare, and initializes the PE array by placing 100000000000 onto the *control-in* port.

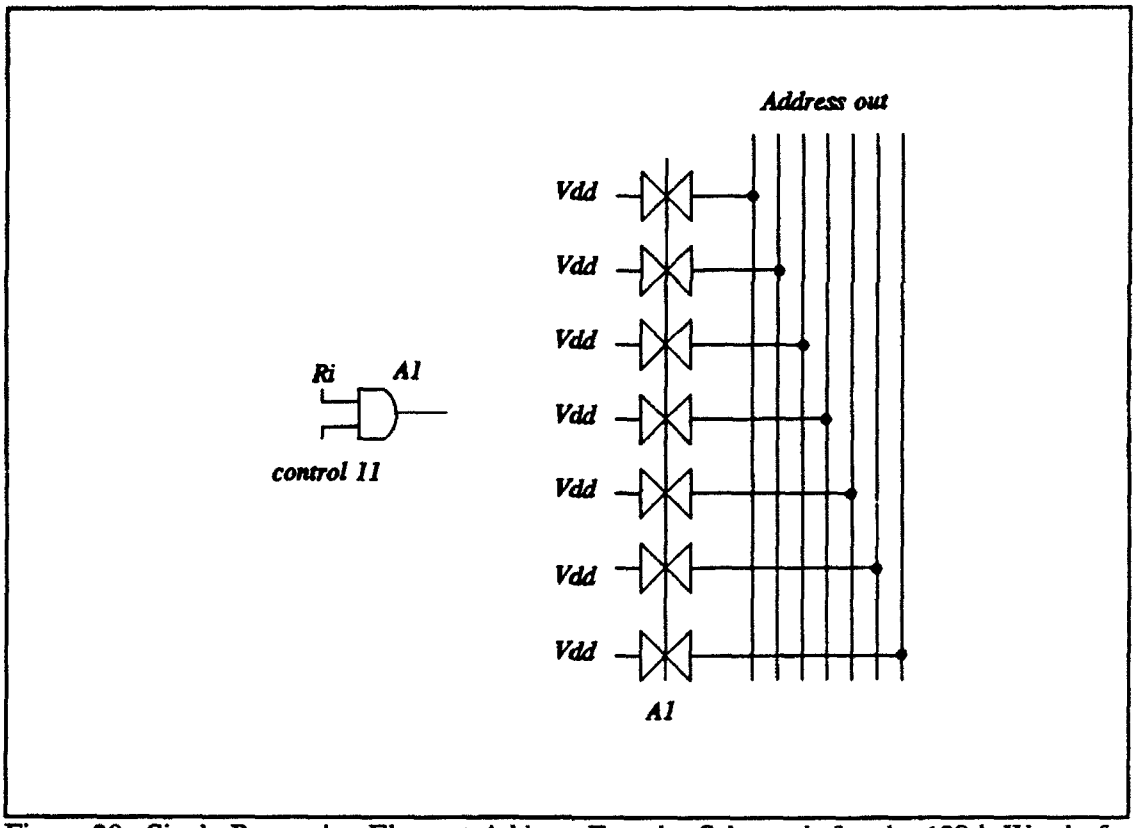

Figure 30. Single Processing Element Address Encoder Schematic for the 128th Word of a 128-Word Memory.

Once sufficient time has passed to decode the *slice-address-in* stimulus, to precharge the slice address-decode circuit, and to initialize the PE array, the instruction is ready to evaluate the valid-data field. The instruction continues when the host enables the *slice-enable* port, and directs the PE array to load the valid-data field by placing 010000000000 onto the *control-in* port.

Once sufficient time has passed to read the bit-slices and load the data into the PE array, the host can continue the instruction by storing the equality and magnitude comparison using the *control-in* stimulus 0001000000000. Next, the host uses the stored comparison results to disable all invalid words. The *control-in* port stimulus is 000010100000.

Table 48

Encoder Transistor-Gate Dimensions.

ENCODER Gate Width = 2λ	Gate Width (λ)
AND-GATE P	6
AND-GATE N	3
T-GATE P	14
T-GATE N	6

Table 49

Encoder Delays.

Address Encoder	Timing Delays (ns)
4 words	t_r=6.1 t_f=5.1
8 words	t_r=6.2 t_f=5.3
16 words	t_r=6.3 t_f=5.3
32 words	t_r=6.5 t_f=5.5
64 words	t_r=11.2 t_f=8.7
128 words	t_r=21.2 t_f=15.2

After completing the valid-data field comparison, the host directs the PE array to re-initialize the equality and magnitude comparator registers using a *control-in* port stimulus of 000000010000. When *control-in(7)* is enabled, the equality register sets and the magnitude register resets.

Once sufficient time has passed to disable all invalid words, the host loads and stores the first bit-slice(s) of the data field. If *control(2)* is disabled, a less-than comparison is performed; otherwise, a greater-than comparison is performed. Simultaneously, an equality comparison is executed and stored. The load/store process is executed *T(s)* times for the

Table 50

Write Time for the Bit-Serial Word-Parallel Associative-Memory Architectures.

Write	12 bits (ns)	20 bits (ns)	36 bits (ns)
4 words	1.4+1.8+1.5+1.9= 6.6	1.4+2.0+1.5+2.0= 6.9	1.4+2.5+1.5+2.3= 7.7
8 words	1.5+1.8+1.5+1.9= 6.7	1.5+2.1+1.5+2.0= 7.1	1.5+2.5+1.5+2.3= 7.8
16 words	1.6+1.8+1.5+1.9= 6.8	1.6+2.1+1.5+2.0= 7.2	1.6+2.5+1.5+2.3= 7.9
32 words	1.9+1.8+1.5+1.8= 7.0	1.9+2.1+1.5+2.0= 7.5	1.9+2.6+1.5+2.3= 8.3
64 words	2.6+1.8+1.5+1.8= 7.8	2.6+2.1+1.5+2.0= 8.2	2.6+2.6+1.5+2.3= 9.2
128 words	4.7+2.0+1.5+1.8= 10.0	4.7+2.1+1.5+2.0= 10.3	4.7+2.6+1.5+2.3= 11.1

Table 51

Read Time for the Bit-Serial Word-Parallel Associative-Memory Architectures.

Read	12 bits (ns)	20 bits (ns)	36 bits (ns)
4 words	1.4+3.1+1.9 = 6.4	1.4+3.4+2.0 = 6.8	1.4+4.0+2.3 = 7.7
8 words	1.5+3.4+1.9 = 6.8	1.5+3.8+2.0 = 7.3	1.5+4.5+2.3 = 8.3
16 words	1.6+4.2+1.9 = 7.7	1.6+4.6+2.0 = 8.2	1.6+5.3+2.3 = 9.2
32 words	1.9+5.3+1.8 = 9.0	1.9+5.8+2.0 = 9.7	1.9+6.7+2.3 = 10.9
64 words	2.6+7.3+1.8 = 11.7	2.6+7.9+2.0 = 12.5	2.6+8.7+2.3 = 13.6
128 words	4.7+11.0+1.8 = 17.5	4.7+11.6+2.0 = 18.3	4.7+12.2+2.3 = 19.2

single BSWPAM architecture, $\lceil T(s)/2 \rceil$ times for the two BSWPAM architecture, and $\lceil T(s)/4 \rceil$ times for the four BSWPAM architecture.

After evaluating the data field, the host directs the PE array to pass the search to the word-select register. When *control-in(5)* is disabled, either $Op_{=}(C(s), db(s))$,

$Op_<(C(s), db(s))$, or $Op_>(C(s), db(s))$, can be used to disable the word-search register; otherwise, either $Op_\neq(C(s), db(s))$, $Op_\leq(C(s), db(s))$, or $Op_\geq(C(s), db(s))$ can be used to disable the register.

Once the results of the search operation are formed, they are passed to the address encoder. The address encoder forms the address of the first word that satisfies the associative-search operation criteria. The host can read this address, reset the associated processing element, then read the address of the next word that satisfies the associative-search operation criteria. This process can continue until all the addresses of words satisfying the associative-search operation criteria are identified. Table 49 gives the encoder delays.

To read the second matching address, the host must query the processing element results. The time to query the PE array depends upon the size of the bit-serial array. The worst case time to identify a matching address occurs when the first and last word in bit-serial array match. For the single BSWPAM architecture, this time is approximately 1.0 ns times *I* plus the time to encode the address. For the two BSWPAM architecture, the time is approximately 0.95 times *I* plus the time to encode the address. For the four BSWPAM architecture, the time is 1.3 times *I* plus the time to encode the address. The single and two BSWPAM have approximately the same delays, while the four BSWPAM is longer. This is attributed to differences in the PE design and the layout height.

5.4.2 Phrase Mode. The phrase mode uses the same features as discussed in the word mode with additional features to establish the word position in a phrase. Like the CAM, the phrase mode uses physical locality to associate words within a phrase, but unlike the CAM, access to words is simpler. The phrase field is used in the phrase mode to identify the word position within a phrase.

The phrase mode write instruction is executed in the same manner as in the word mode, except the phrase field must be preserved. A write instruction must rewrite the

appropriate valid-data and phrase field. Likewise, $Op_{=}(C(s), db(s))$, $Op_{<}(C(s), db(s))$, $Op_{>}(C(s), db(s))$, $Op_{\neq}(C(s), db(s))$, $Op_{\leq}(C(s), db(s))$, and $Op_{\geq}(C(s), db(s))$ are executed similarly except the initial valid-data field search includes a phrase field search. For the single BSWPAM architecture, the phrase field adds three load/store operations to the time to execute an associative-search operation. Since the two BSWPAM architecture evaluates the first bit of the phrase field while evaluating the valid-data field, a single additional load/store operation is necessary to evaluate the two remaining phrase field bits, and since the four BSWPAM architecture evaluates the phrase field while evaluating the valid-data field, a no additional load/store operations are necessary.

The major difference between the word and phrase mode occurs when passing the results of an associative-search operation from one word in a phrase to the next word. Each *PE(i)* output is interconnected to *PE(i-1)* and *PE(i+1)*. The host can transfer the results of an associative-search operation from *PE(i)* to either *PE(i-1)* using a *control-in* stimulus of 000000000100 or *PE(i+1)* using 000000001100. Once passed to the appropriate word and after the processing elements are partially re-initialized, the next associative-search operation can be performed. This implies that the output of *PE(i)* will be disabled, while either *PE(i-1)* or *PE(i+1)* may remain enabled. At the end of searching the phrase, at most one word per phrase will be left enabled. The host will pass the search results of this word to the word within the phrase that contains the data to be read.

5.5 Operation Execution-Time Results

This section combines the data collected in Section 5.4 to estimate the associative-search operation execution time. An associative-search instruction requires the same amount of time whether $OP_{=}(C(s), db(s))$, $Op_{\neq}(C(s), db(s))$, $Op_{<}(C(s), db(s))$, $Op_{>}(C(s), db(s))$, $Op_{\leq}(C(s), db(s))$, or $Op_{\geq}(C(s), db(s))$. Each instruction begins by reading a bit-slice of the

bit-serial array, then storing the bit-slice into the PE array. Then while the PE array evaluates the data, the host reads the next bit-slice, thus pipelining the operation to a certain degree.

Table 52 gives the time necessary to access, read, and store a slice of the bit-serial array. The delays are calculated by summing the maximum value of the slice-address and slice-$\overline{\text{address}}$ delays found in Table 39, the slice-read delay found in Tables 43 through 45, and the load delay. The load delay was measured to be approximately 4.2 ns for each array. For all three circuits, the delays increase linearly. The time for each BSWPAM PE array to evaluate the input is given in Table 53. These delays are calculated by summing the control delays and the processing element search delays. The control delays were not easily accessible. The loads were measured and compared to more accessible delays. The address delay was selected as the most comparable. The difference between the load delays associated with the single BSWPAM architecture and the two and four BSWPAM architectures is attributed to the difference in the number of gates input data must traverse to switch the reference point. Figure 27 shows that the single BSWPAM architecture must traverse two gates to switch S25, while the two and four BSWPAM must traverse four gates to switch S45 and S60, respectively.

The associative-search execution times were evaluated for two scenarios. First, the processing element bit-slice evaluation included the time to store data into the equality register, then into the word-search register, and finally included the time to re-initialize the processing elements. Under this scenario, the processing element bit-slice evaluation delay requires more time than the bit-slice read and load delays. Also, the number of bit-slice evaluations were minimized.

Under the second scenario, the processing element bit-slice evaluation included the time to store data into the equality register and word-search register. The time to re-initialize

Table 52

Bit-Serial Word-Parallel Associative-Memory
Slice Read and Store Times.

Read	12 bits (ns)	20 bits (ns)	36 bits (ns)
Single Bit-Serial Word-Parallel Associative Memory			
4 words	1.9+3.5+4.2 = 9.6	1.9+3.8+4.2 = 9.9	2.9+5.1+4.2 =12.2
8 words	1.9+3.8+4.2 = 9.9	1.9+4.1+4.2 =10.2	2.9+5.5+4.2 =12.6
16 words	1.9+4.3+4.2 =10.4	1.9+4.6+4.2 =10.7	2.9+5.9+4.2 =13.0
32 words	1.9+5.4+4.2 =11.5	1.9+5.7+4.2 =11.8	2.9+7.2+4.2 =14.3
64 words	1.9+7.2+4.2 =13.3	1.9+7.7+4.2 =13.8	2.9+9.4+4.2 =16.5
128 words	1.9+10.4+4.2 =16.5	1.9+11.2+4.2 =17.3	2.9+13.3+4.2 =20.4
Two Bit-Serial Word-Parallel Associative Memory			
4 words	1.8+3.2+4.2 = 9.2	2.2+3.7+4.2 =10.1	3.1+4.6+4.2 =11.9
8 words	1.8+3.6+4.2 = 9.6	2.2+4.0+4.2 =10.4	3.1+5.0+4.2 =12.3
16 words	1.8+4.2+4.2 =10.2	2.2+4.7+4.2 =11.1	3.1+5.7+4.2 =13.0
32 words	1.8+5.3+4.2 =11.3	2.2+5.9+4.2 =12.3	3.1+7.0+4.2 =14.3
64 words	1.8+7.2+4.2 =13.2	2.2+8.0+4.2 =14.4	3.1+9.4+4.2 =16.7
128 words	1.8+10.6+4.2 =16.6	2.2+11.8+4.2 =18.2	3.1+13.6+4.2 =20.9
Four Bit-Serial Word-Parallel Associative Memory			
4 words	1.9+3.7+4.2 = 9.8	2.2+3.8+4.2 =10.2	3.1+4.6+4.2 =11.9
8 words	1.9+4.1+4.2 =10.2	2.2+4.3+4.2 =10.7	3.1+5.2+4.2 =12.5
16 words	1.9+5.0+4.2 =11.1	2.2+5.2+4.2 =11.6	3.1+6.2+4.2 =13.5
32 words	1.9+6.4+4.2 =12.5	2.2+6.7+4.2 =13.1	3.1+7.9+4.2 =15.2
64 words	1.9+9.1+4.2 =15.2	2.2+9.5+4.2 =15.9	3.1+11.0+4.2 =18.3
128 words	1.9+13.5+4.2 =19.6	2.2+14.3+4.2 =20.7	3.1+16.5+4.2 =23.8

requires an additional bit-slice evaluation per initialization for a total of two additional evaluations. Under this scenario, the time to read a bit-slice and load the data is shorter than the processing element bit-slice evaluation time. This scenario requires a maximum of two

Table 53

Bit-Serial Word-Parallel Associative-Memory
Processing-Element Array Search Delays.

Words	Load (ns)	Compare Store (ns)	Search Store (ns)	Partial Initialization (ns)	Global Initialization (ns)	Local Full Initialization (ns)	Search Transfer (ns)
Single Bit-Serial Word-Parallel Associative Memory							
4	5.6	5.9	2.7	6.6	6.6	6.6	3.2
8	5.7	6.0	2.8	6.7	6.7	6.7	3.3
16	5.8	6.1	2.9	6.8	6.8	6.8	3.4
32	6.1	6.4	3.2	7.1	7.1	7.1	3.7
64	6.8	7.1	3.9	7.8	7.8	7.8	4.4
128	8.9	9.2	6.0	9.9	9.9	9.9	6.5
Two Bit-Serial Word-Parallel Associative Memory							
4	6.7	5.9	2.8	5.7	5.7	5.7	2.5
8	6.8	6.0	2.9	5.8	5.8	5.8	2.6
16	6.9	6.1	3.0	5.9	5.9	5.9	2.7
32	7.2	6.4	3.3	6.2	6.2	6.2	3.0
64	7.9	7.1	4.0	6.9	6.9	6.9	3.7
128	10.0	9.2	6.1	9.0	9.0	9.0	5.8
Four Bit-Serial Word-Parallel Associative Memory							
4	6.6	5.9	2.9	6.8	6.8	6.8	3.5
8	6.7	6.0	3.0	6.9	6.9	6.9	3.6
16	6.8	6.1	3.1	7.0	7.0	7.0	3.7
32	7.1	6.4	3.4	7.3	7.3	7.3	4.0
64	7.8	7.1	4.1	8.0	8.0	8.0	4.7
128	9.9	9.2	6.2	10.1	10.1	10.1	6.8

additional bit-slice evaluations.

Both scenarios were evaluated to determine which is the shortest in executing associative-search operations on 8, 16, and 32-cell data fields, and 4, 8, 16, 32, 64, and 128-word bit-serial array lengths. The evaluation determined the second scenario was the shortest for most bit-serial array sizes. Since the second scenario is selected, a least-square analysis of the slice read and store time offsets was performed to develop an equation for each bit-serial array. Equation (53) gives the approximate value of τ_{c1}.

$$\tau_{c1} = (7.96 + 0.11(T(s) + T_{wdf} + T_{pf}) + 0.06l)\ ns \tag{53}$$

Equation (54) gives the approximate value of τ_{c2}.

$$\tau_{c2} = (7.86 + 0.11(T(s) + T_{wdf} + T_{pf}) + 0.07l)\ ns \tag{54}$$

Equation (55) gives the approximate value of τ_{c4}.

$$\tau_{c4} = (8.54 + 0.09(T(s) + T_{wdf} + T_{pf}) + 0.08l)\ ns \tag{55}$$

5.6 Summary and Conclusion

This chapter presented three architectural variations of the BSWPAM organization: the single, two, and four BSWPAM architecture. The critical-path delays of each architecture were identified and, from simulations, measured to develop characteristic equations approximating the instruction-set execution times.

Several counter intuitive results were observed during this aspect of the research. From the beginning, the research realized that the algorithm to execute instructions on the four BSWPAM architecture requires less execution steps than either the single or the two BSWPAM architectures. The research assumed that the four BSWPAM architecture would also be somewhat slower in the execution of each step of the algorithm. This latter

assumption proved to be in error for two reasons. First, the time to read a slice of the bit-serial array is relatively constant for the three versions of the organization. This fact is attributed to an increasing delay to decode the bit-slice address for an increasing bit-vector length and a decreasing delay to form *slice-out*. Second, the number of gates at a particular gate level within the processing element increases from the single to the four BSWPAM architecture, but not the number of gate levels. This implies that the PE delays are fairly constant.

VI. Word-Organized Word-Parallel Extreme-Search Associative Memory

6.1 Introduction

This chapter presents a new associative-memory organization, entitled the word-organized word-parallel extreme-search associative memory (ESAM). This associative-memory organization was designed to demonstrate a processing granularity resembling BPD, RCD operations.

Like a CAM organization, the ESAM organization is word organized, but since the search is bit-position dependent, each bit in a word cannot be compared simultaneously, but rather in a serial fashion, resembling a BSWPAM organization. The term, extreme search, implies comparing the content of the memory array to itself to find either the minima or maxima. The circuitry to execute this comparison makes the ESAM organization unique to any previously discussed design.

The chapter develops the characteristic equations to approximate the read, write, and the associative-search instruction execution times. It begins by describing the organization. It describes the architecture to the transistor level and identifies the critical-path delays. It explains the procedures to execute write, read, and associative-search instructions, and concludes by comparing this memory to a similar design.

Throughout the chapter, enhancements to the ESAM architecture are mentioned. These enhancements allow the architecture to execute BPD, RCI operations, and are necessary to compare different associative-memory organizations in their execution of a mix of associative-search operations.

For a complete VHDL description of the ESAM, refer to Appendix G.

6.2 ESAM Organization and General Operation Description

The ESAM and the CAM share the same organization, though the ESAM array functions differently than the CAM array. The ESAM array consists of a two-dimensional array of ESAM cells with each cell containing the logic to execute its portion of the BPD, RCD and BPI, RCI instructions. Like the CAM organization, the ESAM organization combines words into phrases. Both the word and phrase modes require a single-bit valid-data field, and the phrase mode requires a three-bit phrase field.

The ESAM organization can execute the same instruction set as the CAM organization with eight additional BPD, RCD instructions. To execute a BPD, RCD instruction, the comparison of the most significant comparand bit to the corresponding bit-slice within the ESAM array is used to initiate the comparison of the second most significant comparand bit to the corresponding bit-slice, which is used to initiate the comparison of the third most significant comparand bit to the corresponding bit-slice, etc. The comparison results of the least significant comparand bit to the corresponding bit-slice of the ESAM array will identify words that match the search criteria and are stored in the search-results register.

6.3 ESAM Architecture Description

The CAM and ESAM architectures are similar, using similar components and port names. The ESAM architecture consists of the data-write circuit, the data-read circuit, the word-select circuit, and the ESAM array. The architecture interfaces with the host using six ports: *data-in*, *mask-in*, *write*, *data-out*, *control-in*, and *control1-in*. Chapter IV defined the first five ports. *Control1-in* is used to choose either $Op_{=}((C(s), db(s))$, $Op_{min}(db(s))$, or $Op_{max}(db(s))$.

Within this chapter, each circuit description identifies and measures potential critical-paths delays for simulated word lengths of N equal to 12, 20, and 36 cells and I equal to 8, 16, 32, 64, and 128 words. Insufficient computer memory prohibited the simulation of the entire architecture. The architecture simulations require a single model. The ESAM array is modified to simulate three ESAM words with their ports loaded to represent the array. Two precharge circuits were necessary for the maxima and minima ports, one for word lengths less than or equal to 32, and the other for word lengths greater than 32. The precharge circuits appear to not affect the match delay slopes. A single phrase in the word-select circuit is simulated. The other phrases are replaced by a circuit that represents the critical path-delay circuitry within the phrase. When simulations mandated a static capacitance representation of a dynamically changing capacitive value, the worst case scenario was chosen.

6.3.1 Data-Write Circuit Description. The ESAM architecture uses the data-write circuit found in the CAM architecture description. The transistor-gate dimensions are equal, though as Table 54 shows, the layout dimensions are different. Table 54 also gives the layout dimensions for the other circuits discussed within this chapter.

Table 55 gives the data, $\overline{data}$, mask, and data precharge delays. The data, $\overline{data}$, and data precharge delays are defined in the SRAM architecture description. The mask delay is defined in the CAM architecture description. These delays linearly increase as the number of words in the ESAM array linearly increases. The load on *data(n)*, $\overline{data}(n)$, and *mask(n)* are 15, 16, and 27 fF/word, respectively. Since these loads are approximately the same as the CAM cell produced, the delays are approximately equal to the corresponding CAM delays. Table 56 provides the least-square analysis of the delays which when compared to Table 18 shows that the data and $\overline{data}$ delays are approximately equal. The table also provides the least-square analysis for other critical-path delays presented in this section.

Table 54

Extreme-Search Associative-Memory Layout Dimensions.

	Width (λ)	Height (λ)
Control-1 Driver	18	104
Control-2 Driver	32	150
Extreme Memory Cell	64	113
Extreme Select	64	200
Phrase-Select Circuit	799	664
Read Circuit	64	50
Write Circuit	64	387

6.3.2 ESAM-Array Description. The ESAM architecture stores data within the ESAM array. Figure 31 shows the ESAM array. It consists of *I* words with each word containing *N* ESAM cells, identified as *ESAM cell(0, 0)* to *ESAM cell(I-1, N-1)*. It contains *I word-select*, *search-select*, and *word-match* lines and *N data*, $\overline{data}$, and *mask* lines, using the same nomenclature as the CAM. It also contains *N minima*, *maxima*, and *extrema* lines, identified as *minima(0)* to *minima(N-1)*, *maxima(0)* to *maxima(N-1)*, and *extrema(0)* to *extrema(N-1)*, respectively.

Unlike any other memory array previously discussed, the ESAM array has built-in control circuitry to select either $Op_{=}((C(s), db(s))$, $Op_{min}(db(s))$, or $Op_{max}(db(s))$. The host interfaces with this control circuitry using the *control1* port. When *control1(1)* is disabled, the ESAM array can execute the CAM instruction set. When *control1(0)* is disabled and *control1(1)* is enabled, the ESAM array can execute the search-all-minima and search-subset-minima instructions. When *control1(0)* and *control1(1)* are enabled, the ESAM array can execute the search-all-maxima and search-subset-maxima instructions.

Table 55

Data, $\overline{Data}$, and Data Precharge Delays for the ESAM Architecture.

Extreme Write Circuit	data delay (ns) reference: *write* 50% pt	$\overline{data}$ delay (ns) reference: *write* 50% pt	mask delay (ns) reference: *mask-in* 50% pt range:0 to 5.0V	precharge delay (ns) reference: *write* 50% pt
8 words	0 to 3.9 V t_r<0.5 t_f= 0.0	0 to 3.9 V t_r<0.5 t_f= 0.1	t_r= 0.8 t_f= 1.5	0 to 3.6 V t_r=12.3
16 words	0 to 3.8 V t_r<0.5 t_f= 0.2	0 to 3.8 V t_r<0.5 t_f= 0.2	t_r= 0.9 t_f= 1.6	0 to 3.5 V t_r=14.5
32 words	0 to 3.8 V t_r<0.5 t_f= 0.2	0 to 3.8 V t_r<0.5 t_f= 0.3	t_r= 1.0 t_f= 1.8	0 to 3.4 V t_r= 18.0
64 words	0 to 3.8 V t_r<0.5 t_f= 0.5	0 to 3.8 V t_r<0.5 t_f= 0.4	t_r= 1.3 t_f= 2.2	0 to 3.3 V t_r=24.4
128 words	0 to 3.8 V t_r<0.5 t_f= 0.9	0 to 3.8 V t_r<0.5 t_f= 0.8	t_r= 1.9 t_f= 3.0	0 to 3.0 V t_r=35.6

Figure 32 shows the ESAM-cell memory schematic. Figure 33 shows the ESAM-cell logic schematic for a cell in the first bit-slice of the ESAM array, and Table 57 gives the ESAM-cell transistor-gate dimensions.

A datum is written into and read from the ESAM cell in the same manner as described for the CAM cell. Refer to Figure 31. The ESAM cell executes an equivalence comparison of the cell content with *extrema(n)* continuously. If *search-select(i)* is enabled and a match occurs, then the output enables the next ESAM cell within the word to execute a comparison; otherwise, the output disables the remainder of the words from further comparison.

6.3.3 Data-Read Circuit Description. The ESAM architecture uses the data-read circuit presented in the CAM architecture description, though as Table 54 shows, the layout

Table 56

Least-Square Analysis for the ESAM Architecture.

Delay	Coefficient of Determination	Offset (ns)	Slope (ns/word)
Data\ Data-not Fall Delay	0.99\ 0.99	0.03\ 0.09	0.01\ 0.02
Mask Rise\Fall Delay	0.99\ 0.97	0.64\ 1.38	0.01\ 0.01
Read Rise Delay for 12\ 20\ 36-bit words	0.98\ 0.98\ 0.98	9.25\ 10.37\ 10.39	0.29\ 0.29\ 0.30
Read Fall Delay for 12\ 20\ 36-bit words	0.86\ 0.87\ 0.90	7.21\ 7.35\ 7.41	0.09\ 0.09\ 0.09
Control(4) Rise\Fall Delay	1.00\ 1.00	0.38\ 0.64	0.15\ 0.06
$\overline{\text{Control}}$(4) Rise\ Fall Delay	1.00\ 1.00	0.55\ 0.82	0.14\ 0.07
Control(6) Rise\ Fall Delay	1.00\ 0.79	1.51\ 1.78	0.30\ 0.03
Word-Select Rise Delay for 12\ 20\ 36-bit Words	1.00\ 1.00\ 1.00	4.18\ 4.57\ 5.03	0.30\ 0.29\ 0.30
Word-Select Fall Delay for 12\ 20\ 36-bit words	1.00\ 1.00\ 0.99	2.53\ 2.73\ 3.18	0.05\ 0.05\ 0.05
Search-Select Rise Delay for 12\ 20\ 36-bit words	1.00\ 1.00\ 1.00	4.89\ 5.04\ 5.43	0.30\ 0.30\ 0.30
Search-Select Fall Delay for 12\ 20\ 36-bit words	1.00\ 1.00\ 1.00	2.16\ 2.26\ 2.43	0.03\ 0.03\ 0.03
Word-Match Bit-Slice Evaluation Delay for Equivalence, Minima, & Maxima Operation	1.00\ 1.00\ 1.00	1.5\ 4.4\ 4.8	0.0\ 0.10\ 0.10
Write Delays for 12\20\36-bit words	1.00\ 1.00\1.00	10.11\10.70\11.95	0.34\ 0.34\ 0.34
Read Delays for 12\20\36-bit words	0.99\ 9.98\ 0.99	11.79\ 12.75\ 13.56	0.34\ 0.34\ 0.35
Search-All-Eq; 12\20\36-bit words	1.00\ 1.00\ 1.00	30.33\ 42.48\ 66.88	0.44\ 0.44\ 0.44
Search-All-Min; 12\20\36-bit words	1.00\ 1.00\ 1.00	65.23\100.65\271.5	2.43\ 2.43\ 4.43
Search-All-Max; 12\20\36-bit words	1.00\ 1.00\ 1.00	69.73\108.15\185.1	1.60\ 2.37\ 3.92

dimensions are different. Table 58 gives the read delays. As is seen from the table, the read delays linearly increase as the number of ESAM cells in a word and as the number of words in ESAM array linearly increase. The read fall delay is not externally driven, but rather is a

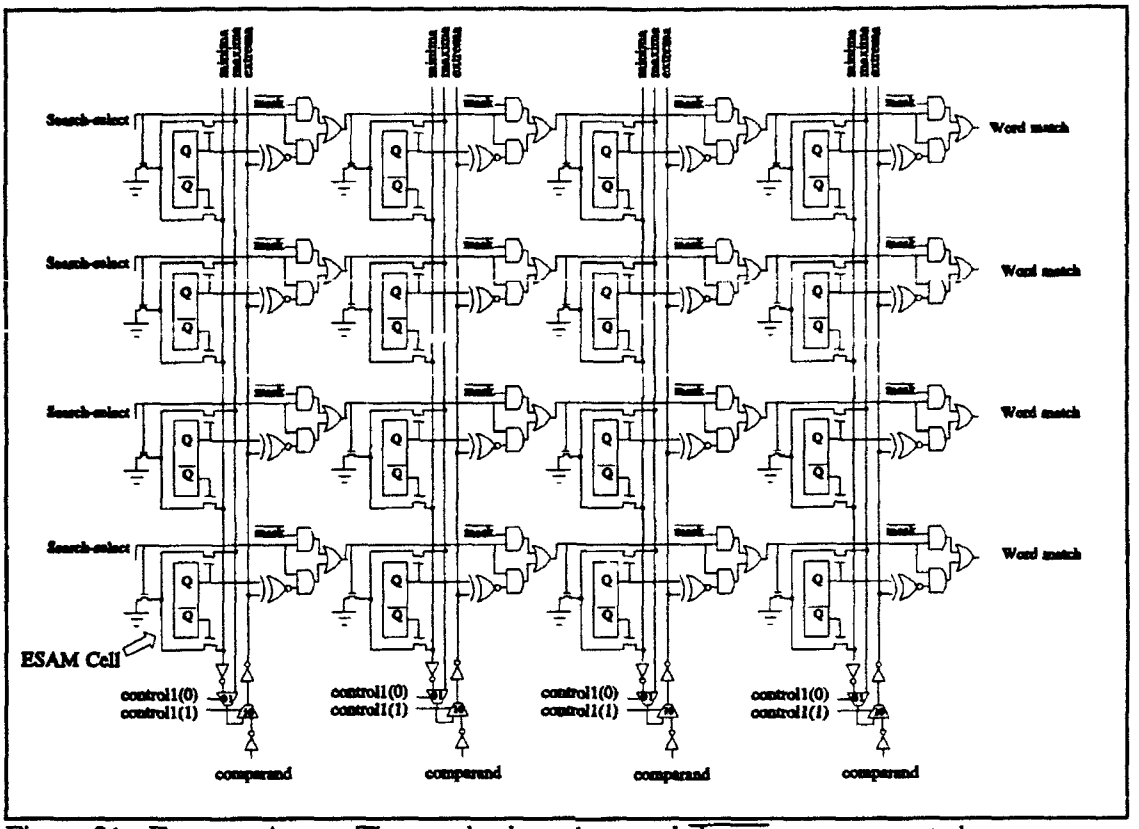

Figure 31. Extreme Array. The word-select, data, and $\overline{data}$ ports are not shown.

function of the data and $\overline{data}$ precharge delays. As the number of words in the ESAM array increases, the precharge voltage range decreases, which causes the read circuit output-voltage range to decrease. The decreased output-voltage range decreases the noise margin and skews the switching times. Since the CAM and ESAM arrays are driven by the same data-write circuit and the loads produced by the CAM and ESAM cells are similar, and both architectures use the same read circuit, the read delays are approximately equal. A comparison of the read delays in Table 56 with the read delays in Table 18 confirms this observation.

6.3.4 Word-Select Circuit Description. The ESAM architecture uses the word-select circuit presented in the CAM architecture description, though the layout dimensions and loads to external circuits are different. Table 54 gives the layout dimensions for the control-1

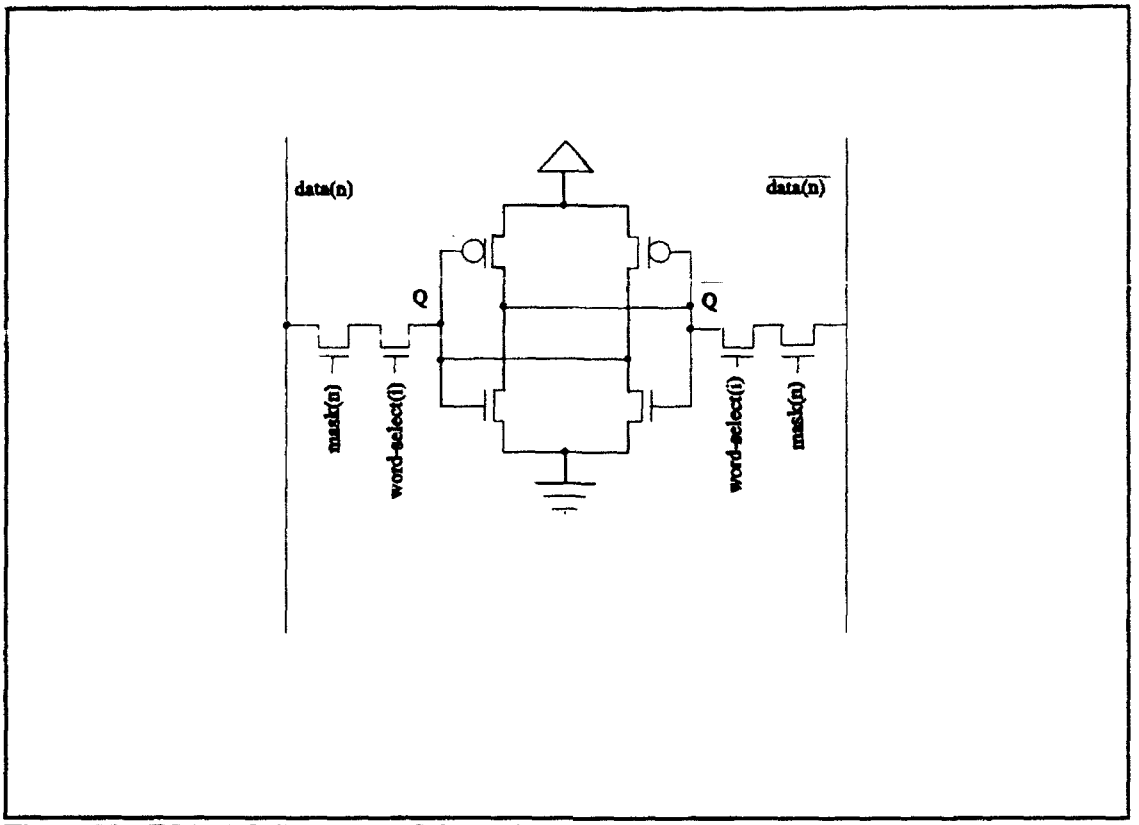

Figure 32. ESAM-Cell Memory Schematic.

Table 57

ESAM-Cell Transistor-Gate Dimensions.

Extrema Cell Gate Length = 2λ	Gate Width (λ)
Memory P-type Transistors	8
Memory N-Type Transistors	3
Logic P-type Transistors	4
Logic N-type Transistors	3

driver, the control-2 driver, and the phrase-select circuit.

Table 59 gives the control delays for simulated ESAM array lengths. The table shows that the control, $\overline{\text{control}}$, and control(4) delays linearly increase with a linear increase in

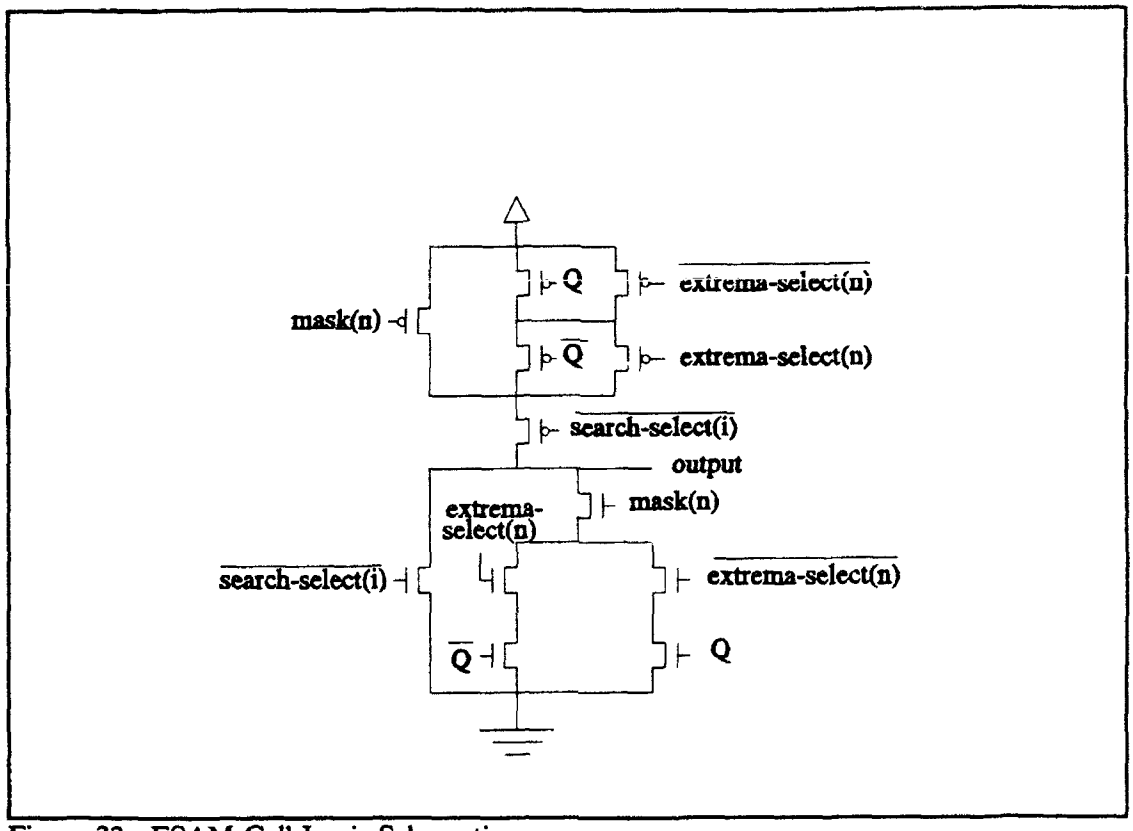

Figure 33. ESAM Cell Logic Schematic

the number of words.

Table 60 gives the delays to drive the *word-select* port. Table 61 gives the delays to drive the *search-select* port. Table 62 gives the delays to drive a bit-slice of the ESAM array for equivalence, minima, and maxima operations. As can be seen from Table 56, the delays to initiate a read/write/search instruction and to drive the appropriate words in ESAM array linearly increase with a linear increase in the number of words. The difference between the equivalence operation offset and slope and the minima and maxima operation offsets and slopes is attributed to a difference in critical paths. The extrema port in each bit-slice of the ESAM array is externally driven during an equivalence operation, though it is internally driven by the more time consuming minima and maxima ports during a minima and maxima operations.

Table 58

Read Delays for the ESAM Architecture.

Read Circuit	read delay (ns) 12 bits ref: *control-in(6)*	read delay (ns) 20 bits ref: *control-in(6)*	read delay (ns) 36 bits ref: *control-in(6)*
8 words	1.7 - 3.8 V t_r=10.1 t_f= 6.3	1.7 - 3.8 V t_r=10.8 t_f= 6.5	1.7 - 3.8 V t_r=11.7 t_f= 6.8
16 words	1.8 - 3.8 V t_r=13.3 t_f= 7.7	1.8 - 3.8 V t_r=14.0 t_f= 7.8	1.8 - 3.8 V t_r=14.5 t_f= 8.3
32 words	1.8 - 3.8 V t_r=20.6 t_f=12.1	1.8 - 3.8 V t_r=21.3 t_f=12.3	2.0 - 3.8 V t_r=19.8 t_f=11.3
64 words	2.0 - 3.7 V t_r=30.8 t_f=15.2	2.0 - 3.7 V t_r=31.4 t_f=15.2	2.0 - 3.7 V t_r=32.4 t_f=15.7
128 words	2.0 - 3.1 V t_r=45.5 t_f=17.9	2.2 - 3.1 V t_r=45.3 t_f=18.1	2.2 - 3.1 V t_r=46.8 t_f=18.2

6.4 ESAM Operation Description

When *control1(1)* is disabled, the ESAM architecture executes write, read, and BPI, RCI instructions in the same manner as discussed in the CAM instruction operation description. When *control1(1)* is enabled, the ESAM architecture can execute the BPD, RCD instructions. These instructions are presented next. First, the write instructions are presented. Second, the read instruction is presented. Finally, the associative-search instructions are presented.

6.4.1 Write Instructions. The write instructions execution procedures are described in the CAM operation description. Like the CAM, the write-all instruction is the most time consuming. The time to execute a write-all instruction is tabulated in Table 63. The first value of each cell in the table is the word-select rise delay found in Table 60. A 3.4 ns delay is added to allow the data to be stored. The third delay is the word-select fall delay. As is

Table 59

Control Delays for the ESAM Architecture.

PE Array	control delay (ns) reference: *control-in(2)* 50% pt range: 0 to 5 V	$\overline{control}$ delay (ns) reference: *control-in(2)* 50% pt range: 0 to 5 V	control(4) delay (ns) reference: *control-in(4)* 50% pt range: 0 to 5 V
8 words	t_r= 1.3 t_f= 1.0	t_r= 1.7 t_f= 1.4	t_r= 1.1 t_f= 1.0
16 words	t_r= 2.5 t_f= 1.6	t_r= 2.8 t_f= 1.9	t_r= 3.4 t_f= 2.6
32 words	t_r= 4.8 t_f= 2.7	t_r= 4.9 t_f= 2.8	t_r= 7.8 t_f= 3.1
64 words	t_r= 9.6 t_f= 4.9	t_r= 9.4 t_f= 5.0	t_r=17.2 t_f= 3.7
128 words	t_r=19.4 t_f= 8 7	t_r=18.3 t_f= 9.2	t_r=36.8 t_f= 4.7

seen from Table 56, the write delays linearly increase as the number of words in the ESAM array linearly increases. Notice that increasing the number of ESAM cells in a word increases the offset, but the slopes are nearly constant. From the least-square analysis of the three offsets, a single equation characterizing the write-all instruction is formed. Equation (56) approximates the ESAM write-all instruction execution time.

$$\tau_{write-all}=(9.08+0.077(T(s)+T_{wdf}+T_{pf})+0.344I)\ ns \tag{56}$$

The percentage difference between the equation and the tabulated data is calculated to be less than 4.8% for all ESAM array sizes.

6.4.2 Read Instruction. The host can select only one read instruction. Its execution is also described in Chapter IV. The time to execute a read instruction is tabulated in Table 64. The first value of each cell in the table is the read delay found in Table 58. The second element is the word-select fall delay. Table 56 shows that the read delays linearly increase as the number of words in the ESAM array and the number of ESAM cells in a word linearly increase. From a least-square analysis of the three offsets, a single equation

Table 60

Word-Select Delays for the ESAM Architecture.

CAM Array	word-select delay (ns) 12 bits reference: *control-in(6)* 50% pt range: 0 to 5 V	word-select delay (ns) 20 bits reference: *control-in(6)* 50% pt range: 0 to 5 V	word-select delay (ns) 36 bits reference: *control-in(6)* 50% pt range: 0 to 5 V
8 word	t_r= 7.1 t_f= 2.8	t_r= 7.5 t_f= 3.0	t_r= 8.3 t_f= 3.4
16 word	t_r= 8.9 t_f= 3.3	t_r= 9.3 t_f= 3.5	t_r=10.1 t_f= 3.9
32 word	t_r=13.2 t_f= 4.2	t_r=13.5 t_f= 4.4	t_r=14.3 t_f= 4.9
64 word	t_r=22.6 t_f= 5.9	t_r=23.0 t_f= 6.1	t_r=23.8 t_f= 6.6
128 word	t_r=42.2 t_f= 8.8	t_r=42.5 t_f= 9.0	t_r=43.3 t_f= 9.4

characterizing the read instruction is formed. Equation (57) approximates the ESAM read instruction execution time.

$$\tau_{read}=(11.096+0.071(T(s)+T_{wdf}+T_{pp})+0.342I)\ ns \qquad (57)$$

The percentage difference between the equation and the tabulated data is calculated to be less than 13.8% for all ESAM array sizes. The increased error as compared to the write instruction is attributed to the data precharge voltage degradation that skews the read delay and voltage range.

6.4.3 Associative-Search Instructions. The host can select one of twelve associative-search instructions: search-all-equal, search-all-not-equal, search-subset-equal, search-subset-not-equal, search-all-minima, search-all-not-minima, search-subset-minima, search-subset-not-minima, search-all-maxima, and search-all-not-maxima, search-subset-maxima, and search-subset-not-maxima. The four equivalence instructions begin when the host places the comparand onto the *data-in* port, identifies the bit-slices involved in the associative-search

Table 61

Search-Select Delays for the ESAM Architecture.

ESAM Array	search-select delay (ns) 12 bits ref: *control-in(6)* range: 0 to 5 V	search-select delay (ns) 20 bits ref: *control-in(6)* range: 0 to 5 V	search-select delay (ns) 36 bits ref: *control-in(6)* range: 0 to 5 V
8 word	t_r= 7.8 t_f= 2.3	t_r= 8.0 t_f= 2.4	t_r= 8.4 t_f= 2.6
16 word	t_r= 9.7 t_f= 2.7	t_r= 9.8 t_f= 2.8	t_r=10.2 t_f= 2.9
32 word	t_r=13.9 t_f= 3.2	t_r=14.1 t_f= 3.3	t_r=14.4 t_f= 3.5
64 word	t_r=23.4 t_f= 4.4	t_r=23.5 t_f= 4.5	t_r=23.9 t_f= 4.7
128 word	t_r=43.1 t_f= 6.2	t_r=43.3 t_f= 6.3	t_r=43.6 t_f= 6.5

instruction using the *mask-in* port, enables the *write* port, and selects the instruction using the *control-in* and *control1-in* ports, without enabling *control-in(6)*. Once sufficient time has passed to form *data*, $\overline{data}$, and *mask*, and to distribute the *control-in* stimulus throughout the word-select circuitry, the host directs the word-select circuit to enable *control-in(6)*. Once the comparison results are stored in the word-select circuit, the host directs the word-select circuit to disable *control-in(6)*, then the *data-in*, *mask-in*, and *write* port stimuli can change. The eight minima and maxima instructions use a similar process, though there is no data or mask stimuli to distribute through the ESAM array.

The search-all-equal instruction represents the longest delay for the equivalence instructions because every word can load the extrema port. The search-all-equal instruction compares the data specified by *data-in* and *mask-in* to every word in the ESAM array. Table 65 gives the delays to execute search-all-equal and search-all-not-equal. The first element in each cell of the table is the search-select delay found in Table 61. The second element is the bit-slice evaluation rise delay found in Table 62, which is multiplied by the

Table 62

Bit-Slice Evaluation Delays for Equivalence, Minima, and Maxima Operations.

ESAM Array Bit-Slice Evaluation Delays	bit-slice evaluation delay (ns) equivalence reference: *search* 50% pt range: 0 to 5.0 V	bit-slice evaluation delay (ns) minima reference: *search* 50% pt range: 0 to 5.0 V	bit-slice evaluation delay (ns) maxima reference: *search* 50% pt range: 0 to 5.0 V
8 word	rise=1.5 fall=1.3	rise=5.0 fall=1.3	rise=5.6 fall=1.2
16 word	rise=1.5 fall=1.3	rise=6.1 fall=1.3	rise=6.5 fall=1.1
32 word	rise=1.5 fall=1.3	rise=7.6 fall=1.3	rise=7.5 fall=1.1
64 word	rise=1.5 fall=1.3	rise=10.9 fall=1.3	rise=11.1 fall=1.1
128 word	rise=1.6 fall=1.3	rise=15.6 fall=1.3	rise=16.5 fall=1.2

number of cells in the word. The third element is the time to store the data into the master portion of a master-slave flip flop found within the word-select circuit. The fourth element is the maximum of the control and $\overline{\text{control}}$ delays found in Table 59. The fifth element is the time to store the search results into the slave portion of the master-slave flip flop. Table 56 shows that the delays linearly increases as the number of words in the ESAM array linearly increases. From a least-square analysis of the three offsets, a single equation characterizing the search-all-equal instruction is formed. Equation (58) approximates the time to execute $Op_{=}((C(s), db(s))$.

$$\tau_{ex}=(12.06+1.50(T(s)+T_{wdf}+T_{pp})+0.45l) \text{ ns for } Op_{=}(C(s),db(s)) \qquad (58)$$

The percentage difference between the equation and the tabulated data is calculated to be less than 1.7% for all ESAM array sizes.

Table 63

Write-All Time for the ESAM Architecture.

Write-All	12 bits (ns)	20 bits (ns)	26 bits (ns)
8 words	7.1 + 3.4 + 2.8 = 13.3	7.5 + 3.4 + 3.0 = 13.9	8.3 + 3.4 + 3.4 = 15.1
16 words	8.9 + 3.4 + 3.3 = 15.6	9.3 + 3.4 + 3.5 = 16.2	10.1 + 3.4 + 3.9 = 17.4
32 words	13.2 + 3.4 + 4.2 = 20.8	13.5 + 3.4 + 4.4 = 21.3	14.3 + 3.4 + 4.9 = 22.6
64 words	22.6 + 3.4 + 5.9 = 31.9	23.0 + 3.4 + 6.1 = 32.5	23.8 + 3.4 + 6.6 = 33.8
128 words	42.2 + 3.4 + 8.8 = 54.4	42.5 + 3.4 + 9.0 = 54.9	43.3 + 3.4 + 9.4 = 56.1

The search-all-minima and search-all-not-minima instructions are executed in the same fashion as search-all-equal, and search-subset-minima and search-subset-not-minima instructions are executed in the same fashion as search-subset-equal except *control-in1(0)* is disabled, *control-in1(1)* is enabled, and *write* is disabled. Table 66 gives the delays to execute search-all-minima, search-all-not-minima, search-subset-minima and search-subset-not-minima. Except the second element, the elements within the table represent the same delays as discussed in the search-all-equal instruction. The second element is the bit-slice evaluation delay for a minima search. The delays linearly increase as the number of words in the ESAM array linearly increases. From the least-square analysis of the three offsets, a single equation characterizing the search-all-minima instruction is formed. Equation (59) approximates the time to execute $Op_{min}(db(s))$.

$$\tau_{ex}=(12.05+4.93(T(s)+T_{wdj}+T_{pp})+0.44I+0.09T(s)I)\ ns\ for\ Op_{min}(C(s),db(s)) \quad (59)$$

Notice that because the minima port depends upon both *I* and *T(s)*, the equation also contains a component dependent upon *I* and *T(s)*. The percentage difference between the equation and the tabulated data is calculated to be less than 7.2% for all ESAM array sizes.

Table 64

Read Time For the ESAM Architecture.

Read	12 bits (ns)	20 bits (ns)	36 bits (ns)
8 words	10.1 + 2.8 = 12.9	10.6 + 3.0 = 13.6	11.7 + 3.4 = 15.1
16 words	13.3 + 3.3 = 16.6	14.0 + 3.5 = 17.5	14.5 + 3.9 = 18.4
32 words	19.5 + 4.2 = 23.7	20.1 + 4.4 = 24.5	19.8 + 4.9 = 24.7
64 words	30.8 + 5.9 = 36.7	31.4 + 6.1 = 37.5	32.4 + 6.6 = 39.0
128 words	45.5 + 8.8 = 54.3	45.3 + 9.0 = 54.3	46.8 + 9.4 = 56.2

The search-all-maxima and search-all-not-maxima instructions are executed in the same fashion as the search-all-equal, and the search-subset-maxima and search-subset-not-maxima instructions are executed in the same fashion as the search-subset-equal instruction except *control-in1* is disabled. Table 67 gives the delays to execute search-all-maxima and search-all-not maxima. The elements in the table are organized in the same manner as the equivalence operation. Table 56 shows that the delays linearly increase as the number of words in the ESAM array linearly increases. From the least-square analysis of the three offsets, a single equation characterizing the search-all-maxima instruction is formed. Equation (60) approximates the time to execute $Op_{max}(db(s))$.

$$\tau_{ex}=(12.93+4.80(T(s)+T_{wdf}+T_{pf})+0.44I+0.1IT(s))\ ns\ for\ Op_{max}(C(s),db(s)) \quad (60)$$

The percentage difference between the equation and the tabulated data is calculated to be less than 3.7% for all ESAM array sizes.

Table 65

$Op_{=}(C(s), db(s))$ Execution Time.

$Op_{=}(C(s), db(s))$	12 bits (ns)	20 bits (ns)	36 bits (ns)
8 words	7.8+(1.5x12)+3.7 +1.7+3.4 = 34.6	8.0+(1.5x20)+3.7 +1.7+3.4 = 46.8	8.4+(1.5x36)+3.7 +1.7+3.4 = 71.2
16 words	9.7+(1.5x12)+3.7 +2.8+3.4 = 37.6	9.8+(1.5x20)+3.7 +2.8+3.4 = 49.7	10.2+(1.5x36)+ 3.7+2.8+3.4 = 74.1
32 words	13.9+(1.5x12)+ 3.7+4.9+3.4 = 43.9	14.1=(1.5x20)+3.7 +4.9+3.4 = 56.1	14.4+(1.5x36)+ 3.7+4.9+3.4 = 80.4
64 words	23.4+(1.5x12)+ 3.7+9.6+3.4 = 58.1	23.5+(1.5x20)+3.7 +9.6+3.4 = 70.2	23.9+(1.5x36)+ 3.7+9.6+3.4 = 94.6
128 words	43.1+(1.6x12)+ 3.7+19.4+3.4 = 88.8	43.3+(1.6x20)+3.7 +19.4+3.4 = 101.2	43.6+(1.6x36)+ 3.7+19.4+3.4 = 127.7

The write-all, read, and search-all equations for $Op_{=}((C(s), db(s))$, $Op_{min}(db(s))$, $Op_{max}(db(s))$ are used in Chapter VII for comparison purposes.

6.5 Comparison of ESAM to ACAM

Since the ESAM is a unique design, other memories of a comparable operation are hard to find. Probably one of the most similar designs is the arithmetic content-addressable memory (ACAM) [20]. The ACAM performs arithmetic operations using a ripple process that is similar to the way the ESAM array propagates data during the equivalence instructions.

Table 66

$Op_{min}(db(s))$ Execution Time.

$Op_{min}(C(s), db(s))$	12 bits (ns)	20 bits (ns)	36 bits (ns)
8 words	7.8+(5.0x12)+3.7+ 1.7+3.4 = 76.6	8.0+(5.0x20)+3.7+ 1.7+3.4 = 116.8	8.4+(5.0x36)+3.7+ 1.7+3.4 = 197.2
16 words	9.7+(6.1x12)+3.7+ 2.8+3.4 = 92.8	9.8+(6.1x20)+3.7+ 2.8+3.4 = 141.7	10.2+(6.1x36)+3.7+ 2.8+3.4 = 239.7
32 words	13.9+(7.6x12)+3.7+ 4.9 +3.4 = 117.1	14.1+(7.6x20)+3.7+ 4.9+3.4 = 178.1	14.4+(7.6x36)+3.7+ 4.9+3.4 = 300.0
64 words	23.4+(10.9x12)+3.7+ 9.6+3.4 = 170.9	23.5+(10.9x20)+3.7+ 9.6+3.4 = 258.2	23.9+(10.9x36)+3.7 +9.6+3.4 = 433.0
128 words	43.1+(15.6x12)+3.7+ 19.4+3.4 = 256.8	43.3+(15.6x20)+3.7+ 19.4+3.4 = 381.8	43.6+(15.6x36)+3.7 +19.4+3.4 = 631.7

Also like the CAM, the ACAM uses a 6-transistor SRAM to store data. It also uses a 14-transistor subtracter, a 4-transistor match circuit, and a 4-transistor borrow-propagate circuit for a total of 28 transistors. The ACAM was designed and evaluated as a 32-bit by 32-word memory array. Simulated results indicate that the ACAM can execute an arithmetic operation on the array in approximately 100 ns. This compares well to the ESAM, which can execute an equivalence operation on the same size array in about 80 ns (as measured from simulations). The ESAM also compares well to the ACAM in size. The ACAM memory cell is 106 by 105 um^2 using 2 um design rules. The ESAM cell is 64 by 113 um^2 using the same design rules. The extreme instructions are not comparable to the ACAM instructions. The ACAM instruction set is record-content independent; therefore, the critical path does not progress through a bit-slice like the extreme operations are required to do in the ESAM architecture.

6.6 Summary and Conclusion

This chapter presented the ESAM. It discussed the purpose of the design, presented a general organization, and its specific architectural implementation. Is discussed how to execute write, read, and associative-search instructions and through simulations calculated the time required to execute each instruction. The execution times were developed into characteristic equations which approximate the instruction-set execution time.

Table 67

$Op_{max}(db(s))$ Execution Time.

$Op_{max}(C(s), db(s))$	12 bits (ns)	20 bits (ns)	36 bits (ns)
8 words	7.8+(5.6x12)+3.7 +1.7+3.4 = 83.8	8.0+(5.6x20)+3.7 +1.7+3.4 =128.8	8.4+(5.6x36)+3.7 +1.7+3.4 = 218.8
16 words	9.7+(6.5x12)+3.7 +2.8+3.4 = 97.6	9.8+(6.5x20)+3.7 +2.8+3.4 =149.7	10.2+(6.5x36)+3.7 +2.8+3.4 = 254.1
32 words	13.9+(7.5x12)+ 3.7+4.9 +3.4 = 115.9	14.1+(7.5x20)+ 3.7+4.9+3.4 = 176.1	14.4+(7.5x36)+3.7 +4.9+3.4 = 296.4
64 words	23.4+(11.1x12)+ 3.7+9.6+3.4 = 173.3	23.5+(11.1x20)+ 3.7+9.6+3.4 = 262.2	23.9+(11.1x36)+ 3.7+9.6+3.4 = 440.2
128 words	43.1+(16.5x12)+ 3.7+19.4+3.4 = 267.6	43.3+(16.5x20)+ 3.7+19.4+3.4 = 399.8	43.6+(16.5x36)+ 3.7+19.4+3.4 = 664.1

The ESAM organization could have been less complicated and faster if the design were to execute only BPD, RCD operations, but the research required a ESAM with features to allow the execution of BPI and BPD, RCI operations. The features include the ability to mask bit-slices from being involved in a search, the ability to select a subset of words to search, the ability to identify matching words, and the ability to identify whether at least one match exists. The architecture was designed with all these features.

VII. Execution-Time and Layout-Dimension Analysis and Results

7.1 Introduction

This chapter combines the quantitative analysis and measured data of the previous chapters to complete the research endeavor. The dissertation proposed two objectives. First, it proposed to develop a decision criteria to select an associative-memory organization that minimizes the execution time of a mix of associative-search operations. Section 7.2 completes this objective. The second objective requires an analysis of the layout dimensions of each architecture to give a designer a rule-of-thumb decision criteria in selecting an organization that does not exceed a predefined layout-dimension limit. Section 7.3 presents this analysis. Section 7.4 combines the execution-time and layout-dimension analyses to determine the best architecture for the execution of write, read, and associative-search instructions and operations.

7.2 Associative-Memory Execution-Time Analysis

This section compares each associative-memory organization in the execution of write, read, and associative-search instructions and operations. The section consists of two analyses. Section 7.2.1 compares each associative-memory architecture to the SRAM in the execution of write and read instructions. Section 7.2.2 evaluates each associative-memory architecture in the execution of a mix of associative-search instructions and operations.

7.2.1 Write and Read-Instruction Comparison. This section compares the write and read-instruction execution times of each architecture discussed within the dissertation. For the reader's convenience, the characteristic equations are restated. Equation (46) is the SRAM write-instruction characteristic equation.

$$\tau_{write} = (3.92+0.042T(s)+0.38I)\ ns \tag{46}$$

Equation (47) is the SRAM read-instruction characteristic equation.

$$\tau_{read} = (3.328+0.041T(s)+0.11I)\ ns \tag{47}$$

Equation (48) is the CAM write-all instruction characteristic equation.

$$\tau_{write\text{-}all} = (10.325+0.079(T(s)+T_{vdf}+T_{pf})+0.233I)\ ns \tag{48}$$

Equation (49) is the CAM read-instruction characteristic equation.

$$\tau_{read} = (11.552+0.101(T(s)+T_{vdf}+T_{pf})+0.376I)\ ns \tag{49}$$

Equation (51) is the BSWPAM write-instruction characteristic equation.

$$\tau_{write} = (5.695+0.051(T(s)+T_{vdf}+T_{pf})+0.028I)\ ns \tag{51}$$

Equation (52) is the BSWPAM read-instruction characteristic equation.

$$\tau_{read} = (5.29+0.066(T(s)+T_{vdf}+T_{pf})+0.091I)\ ns \tag{52}$$

Equation (56) is the ESAM write-all instruction characteristic equation.

$$\tau_{write\text{-}all} = (9.08+0.077(T(s)+T_{vdf}+T_{pf})+0.344I)\ ns \tag{56}$$

Equation (57) is the ESAM read-instruction characteristic equation.

$$\tau_{read} = (11.096+0.071(T(s)+T_{vdf}+T_{pf})+0.342I)\ ns \tag{57}$$

Figures 34 and 35 show a plot of the write and read-instruction execution times for a 32-bit data field. Both plots show a linear increase in the execution time as the number of words in the memory array linearly increases. Not shown, though verified, is the linear increase in the execution time as the number of bits in a word linearly increases.

As the equations reveal, the delays associated with the number of bits in a word are comparable for both instructions and all six architectures. This observation is expected. Each

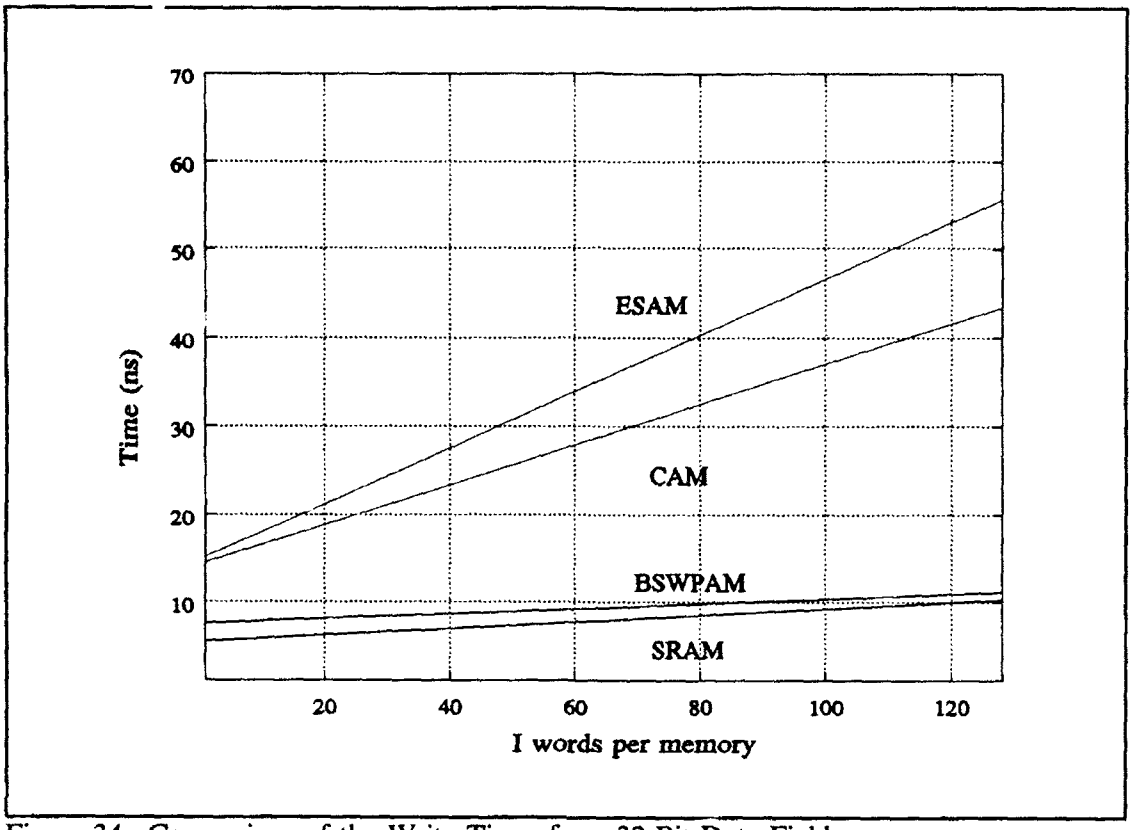

Figure 34. Comparison of the Write Times for a 32-Bit Data Field.

architecture uses a common technique to minimize the polysilicon runs necessary to form the pass transistors for the *word-select(i)* ports. The polysilicon runs are minimized to minimize the corresponding capacitances, which shortens the time necessary to enable or disable the *word-select(i)* ports. Minimizing the polysilicon runs requires using long first level metal runs that consume about twice as much surface area.

The delays associated with the number of words in the memory array are influenced by the way the words are selected. The SRAM and BSWPAM architectures use a similar address-decode circuit; therefore, they have similar delays. The CAM and ESAM architectures use a more complex word-select circuit that requires a longer time for decoding and selecting. These delays are a full order of magnitude greater than the SRAM.

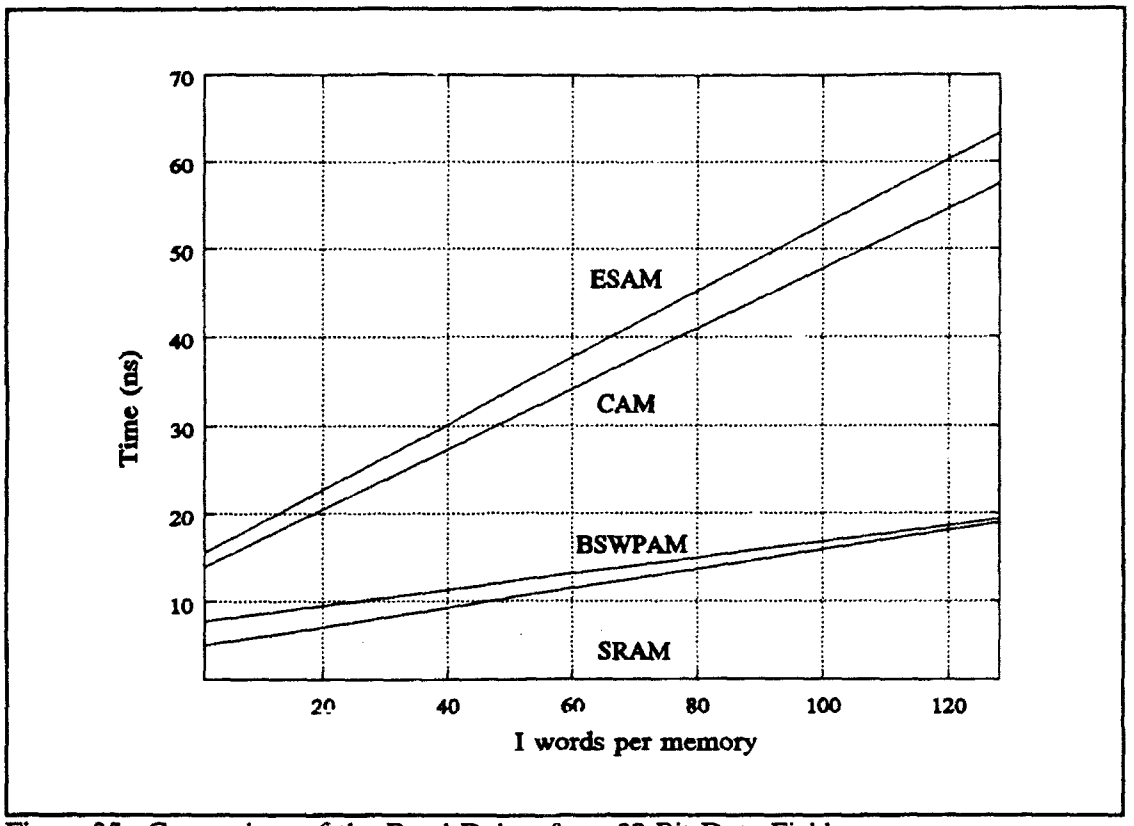

Figure 35. Comparison of the Read Delays for a 32-Bit Data Field.

For each architecture, the execution-time increase associated with increasing the number of words in the memory array exceeds the execution-time increase associated with increasing the number of bits in a word. Therefore, when possible, the memory array should be designed to store a database using a minimum number of words. Ideally, this implies assigning a single record per word.

7.2.2 Associative-Search Comparison. This section compares one architecture to another to confirm the relationship between the hardware-influenced associative-search categories and the associative-memory organizations. It consists of nine comparisons followed by a discussion of the comparison results. The first four comparisons confirm that the CAM architecture executes BPI, RCI operations in a shorter time than each BSWPAM architecture and the ESAM architecture for most memory-array sizes considered. The next two

comparisons confirm that the single BSWPAM architecture executes BPD, RCI operations in a shorter time than the CAM and ESAM architectures for most memory-array sizes. The following two comparisons confirm that the ESAM architecture executes BPD, RCD operations in a shorter time than the CAM for most memory-array sizes and executes BPD, RCD operations in a shorter time than the single BSWPAM architecture for about half of the memory-array sizes. As an example of the way this comparison technique can be used to serve as a decision criteria to select the better associative-memory organization, the last comparison compares the CAM architecture to the single BSWPAM architecture in the word-mode execution of a 50% BPI, RCI and 50% BPD, RCI mix of operations.

The comparisons require data from both the quantitative analysis found in Chapter II and measured data taken from Chapters IV through VI. For convenience these equations are restated. Equation (36) gives the general equation for the CAM architecture execution time for a mix of associative-search operations.

$$\tau_{phrase\ mode}=M(T(s)+(1-T(s))x-(\frac{T(s)+1}{2}-\frac{1}{2^{T(s)}})y)\tau_{cam}+(M-1)\tau_{Tcam}\ ns \qquad (36)$$

Equation (38) gives the equation for the single BSWPAM architecture execution time.

$$\tau_{phrase\ mode}=M(T(s)+T_{wdf}+T_{pf})\tau_{c1}+(M-1)\tau_{T1}\ ns \qquad (38)$$

Equation (40) gives the equation for the two BSWPAM architecture execution time.

$$\tau_{phrase\ mode}=M(\lceil\frac{T(s)}{2}\rceil+\lceil\frac{T_{wdf}+T_{pf}}{2}\rceil+2\lceil\frac{T(s)}{2}\rceil)\tau_{c2}+(M-1)\tau_{T1}\ ns \qquad (40)$$

Equation (42) gives the equation for the four BSWPAM architecture execution time.

$$\tau_{phrase\ mode}=M(\lceil\frac{T(s)}{4}\rceil+\lceil\frac{T_{wdf}+T_{pf}}{4}\rceil+3\lceil\frac{T(s)}{4}\rceil)\tau_{c4}+(M-1)\tau_{T1}\ ns \qquad (42)$$

Equation (45) gives the equation for the ESAM architecture execution time.

$$\tau_{phrase\ mode} = M(1+(\frac{T(s)-3}{2}+\frac{1}{2^{T(s)}})y)\tau_{ex}+(M-1)\tau_{Tex}\ ns \quad (45)$$

Equation (50) gives the approximate value of τ_{cam}.

$$\tau_{cam} \approx 12.966+0.116(T(s)+T_{wdf}+T_{pf})+0.331I)\ ns \quad (50)$$

Equation (53) gives the approximate value of τ_{c1}.

$$\tau_{c1} \approx 7.96+0.11(T(s)+T_{wdf}+T_{pf})+0.06I\ ns \quad (53)$$

Equation (54) gives the approximate value of τ_{c2}.

$$\tau_{c2} \approx 7.86+0.11(T(s)+T_{wdf}+T_{pf})+0.07I\ ns \quad (54)$$

Equation (55) gives the approximate value of τ_{c4}.

$$\tau_{c4} \approx 8.54+0.09(T(s)+T_{wdf}+T_{pf})+0.08I\ ns \quad (55)$$

Equation (58) gives the approximate value of τ_{ex} when executing an equivalence instruction.

$$\tau_{ex} \approx 12.06+1.50(T(s)+T_{wdf}+T_{pf})+0.45I\ ns \quad (58)$$

Equation (60) gives the worst of the two extreme instruction approximations.

$$\tau_{ex} = (12.05+4.93(T(s)+T_{wdf}+T_{pf})+0.44I+0.09IT(s))\ ns\ for\ Op_{max}(C(s),db(s)) \quad (60)$$

Each comparison is performed for both the word and phrase modes. The transfer delays for the BSWPAM architectures are found in Table 53. Though the transfer delays are less than the bit-slice evaluation delays, they are set equal to give a constant execution-step time. The transfer delays for the CAM and ESAM architectures are executed in parallel with other operations and can be considered to be zero, but for this general analysis, the transfer delays are set equal to the equivalence search delays. This decision yields a more conservative result in the following comparisons. T_{wdf} is defined to be a single bit, while T_{pf} is defined to be $\lceil \log_2(M) \rceil$, where M equals one in the word mode and eight in the phrase mode.

7.2.2.1 Comparison of the CAM Architecture to the Single BSWPAM Architecture in the Execution of BPI, RCI Operations. The comparison process begins by calculating the curve that delineates the regions where either the CAM or the single BSWPAM architecture is the better selection. The delineating curve is defined by setting Equations (36) and (38) equal. τ_{cam} is set to τ_{Tcam}, and τ_{cl} is set to τ_{Tl}. Equations (50) and (53) define τ_{cam}, and τ_{cl}, respectively. The value of x is set to 1, y is set to 0, and z is set to 0, to denote a BPI, RCI operation. This equality is mathematically expressed by

$$\begin{aligned}(2M-1)(12.966+0.116(T(s)+T_{wdf}+T_{pf})+0.331I) = \\ (M(T(s)+T_{wdf}+T_{pf})+M-1)(7.96+0.11(T(s)+T_{wdf}+T_{pf})+0.06I)\end{aligned} \quad (61)$$

When the architectures operate in the word mode, M equals 1, T_{wdf} equals 1, and T_{pf} equals 0. Equation (61) reduces to

$$5.012-8.064T(s)-0.11T(s)^2 = 0.06(T(s)-4.517)I \quad (62)$$

The equation shows that I is positive when $T(s)$ is less than five and is negative otherwise. When the architectures operate in the phrase mode, M equals 8, which implies T_{pf} equals 3. In this case, Equation (61) reduces to

$$-126.15-69.75T(s)-0.88T(s)^2 = 0.48(T(s)-5.47)I \quad (63)$$

The equation shows that I is positive when $T(s)$ is less than six and is negative otherwise.

Figure 36 shows the comparison. The region to the left of each delineating curve denotes the memory-array sizes where the single BSWPAM architecture is the better selection. Likewise, the region to the right of each curve shows the memory-array sizes where the CAM architecture is the better selection. The figure reveals that for data-field lengths greater than six bits, the CAM is the better selection; otherwise, the single BSWPAM may be the better selection depending upon the mode and the memory array size. The word-mode and phrase-mode curves converge as the number of words in the memory array

increases. This implies that the overhead associated with transferring data between words within a phrase becomes a small portion of the total delay as I becomes large.

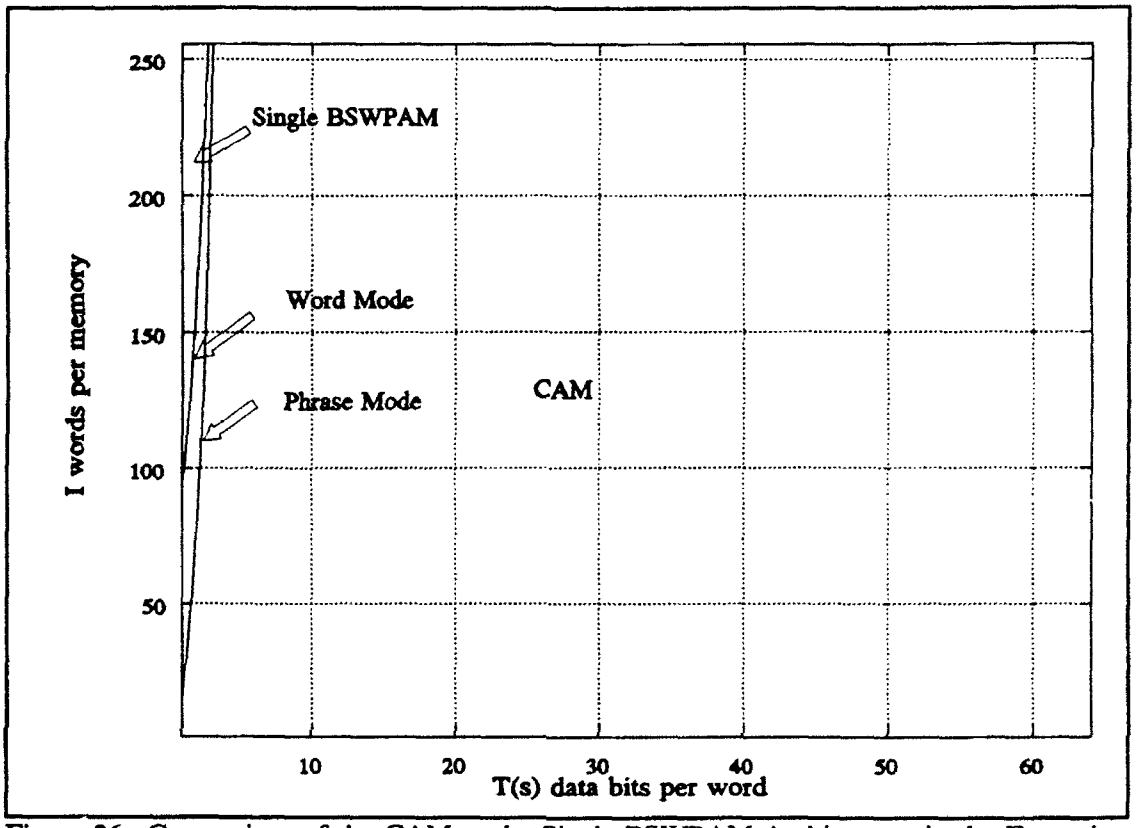

Figure 36. Comparison of the CAM to the Single BSWPAM Architecture in the Execution of BPI, RCI Operations.

7.2.2.2 Comparison of the CAM Architecture to the Two BSWPAM Architecture in the Execution of BPI, RCI Operations. To calculate the delineating curve identifying the memory-array sizes where the CAM architecture or the two BSWPAM architecture is the better selection, let the CAM architecture execution time, denoted by Equation (36), equal the two BSWPAM architecture execution time, denoted by Equation (40). The equation is mathematically expressed by

$$(2M-1)(12.966+0.116(T(s)+T_{vdf}+T_{pf})+0.331I) =$$
$$(M(\lceil\frac{T(s)}{2}\rceil+\lceil\frac{T_{vdf}+T_{pf}}{2}\rceil)+M-1)(7.86+0.11(T(s)+T_{vdf}+T_{pf})+0.07I) \quad (64)$$

When the architectures operate in the word mode, Equation (64) reduces to

$$5.112+0.006T(s)-7.97\lceil\frac{T(S)}{2}\rceil-0.11T(s)\lceil\frac{T(s)}{2}\rceil = 0.07(\lceil\frac{T(s)}{2}\rceil-3.729)I \quad (65)$$

The equation shows that I is positive when $\lceil T(s)/2 \rceil$ is less than four and is negative otherwise. When the architectures operate in the phrase mode, Equation (64) reduces to

$$10.55-0.79T(s)-66.4\lceil\frac{T(s)}{2}\rceil-0.88T(s)\lceil\frac{T(s)}{2}\rceil = 0.56I(\lceil\frac{T(s)}{2}\rceil-5.99) \quad (66)$$

The equation shows that I is positive when $\lceil T(s)/2 \rceil$ is less than six and is negative otherwise.

Figure 37 shows the comparison of the CAM architecture to the two BSWPAM architecture in the execution of BPI, RCI operations. The figure, along with Equation (66) reveals that for data-field lengths greater than thirteen bits, the CAM is the better selection; otherwise, the two BSWPAM may be the better selection depending upon the mode and the memory-array size. The two BSWPAM architecture uses an improved algorithm to execute BPI, RCI operations as compared to the single BSWPAM architecture. Also, τ_{c1} approximately equals τ_{c2}, so the total time the two BSWPAM architecture requires to execute a BPI, RCI operation is approximately one-half the time required by the single BSWPAM architecture. This observation explains the improved performance of the two BSWPAM architecture over the single BSWPAM architecture when compared to the CAM architecture. The curves are not exact representations of the equations. The ceiling function is smoothed to allow plotting. This technique is used for any future comparison using the two and four BSWPAM architectures.

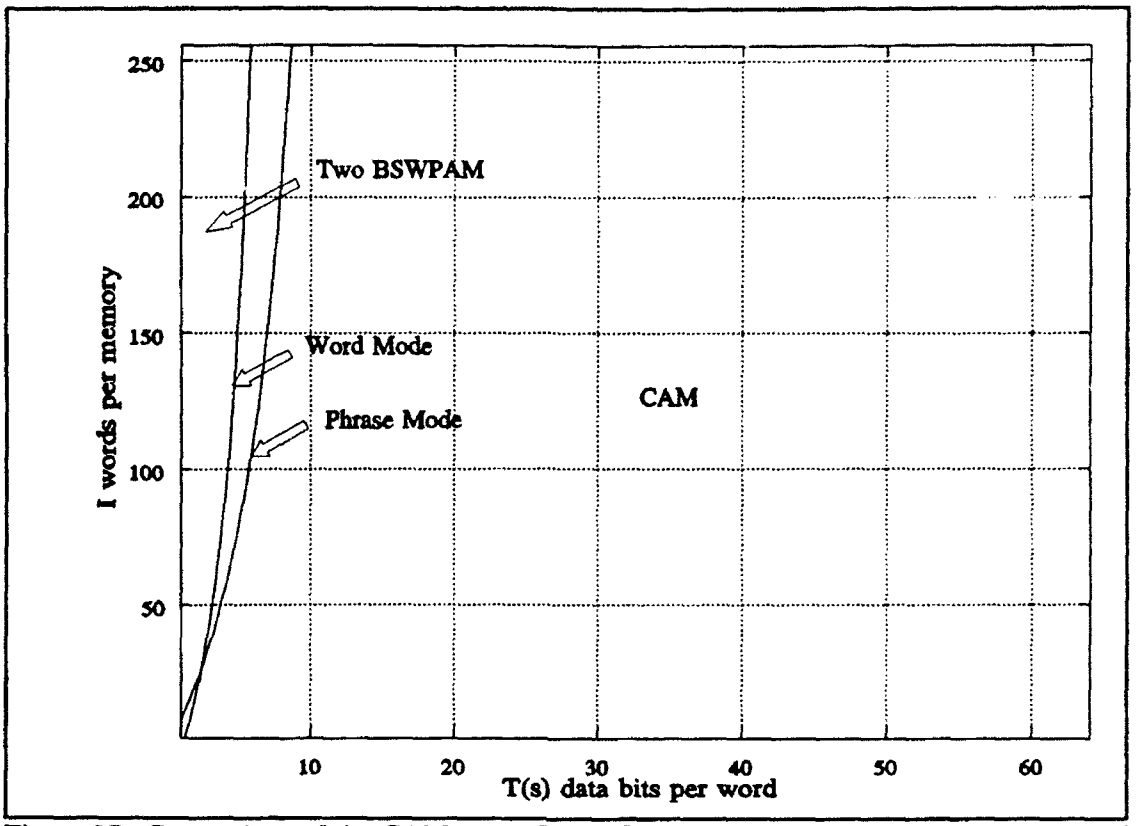

Figure 37. Comparison of the CAM to the Two BSWPAM architecture in the Execution of BPI, RCI Operations.

7.2.2.3 Comparison of the CAM Architecture to the Four BSWPAM Architecture in the Execution of BPI, RCI Operations. Let the time a CAM requires to execute a BPI, RCI operation, denoted by Equation (36), equal the time required by the four BSWPAM architecture, denoted by Equation (42). The equation is fulfilled by the mathematical expression

$$(2M-1)(12.966+0.116(T(s)+T_{wdf}+T_{pf})+0.331I) = \\ (M(\lceil\frac{T(s)}{4}\rceil+\lceil\frac{T_{wdf}+T_{pf}}{4}\rceil)+M-1)(8.54+0.09(T(s)+T_{wdf}+T_{pf})+0.08I) \quad (67)$$

When the architectures operate in the word mode, Equation (67) reduces to

$$4.452+0.026T(s)-8.63\lceil\frac{T(s)}{4}\rceil-0.09T(s)\lceil\frac{T(s)}{4}\rceil = 0.08(\lceil\frac{T(s)}{4}\rceil-3.138)I \qquad (68)$$

The equation shows that I is positive when $\lceil T(s)/4 \rceil$ is less than four and is negative otherwise. When the architectures operate in the phrase mode, Equation (67) reduces to

$$67.95+0.39T(s)-71.2\lceil\frac{T(s)}{4}\rceil-0.72T(s)\lceil\frac{T(s)}{4}\rceil = 0.64(\lceil\frac{T(s)}{4}\rceil-5.88) \qquad (69)$$

The equation shows that I is positive when $\lceil T(s)/4 \rceil$ is less than six and is negative otherwise.

Figure 38, in conjunction with Equations (68) and (69), shows the comparison of the CAM architecture to the four BSWPAM architecture in the execution of BPI, RCI operations. The figure and equations reveal that for word lengths greater than 20 bits, the CAM is the better selection; otherwise, the four BSWPAM may be the better selection depending upon the mode and the memory-array size. The four BSWPAM architecture uses an improved algorithm to execute BPI, RCI operation as compared to either the single or two BSWPAM architecture. Also, τ_{c4} approximately equals τ_{c1} and τ_{c2}, so the total time to execute a BPI, RCI operation is approximately one-fourth of time required by the single BSWPAM architecture. This observation explains the improved performance of the four BSWPAM architecture over the single and two BSWPAM architectures when compared to the CAM architecture.

7.2.2.4 Comparison of the CAM Architecture to the ESAM Architecture in the Execution of BPI, RCI Operations. Let the time a CAM requires to execute a BPI, RCI operation, denoted by Equation (36), equal the time required by the ESAM architecture denoted by Equation (45). The equality is fulfilled by the mathematical expression

$$12.966+0.116(T(s)+T_{wdf}+T_{pf})+0.331I = 12.06+1.50(T(s)+T_{wdf}+T_{pf})+0.45I \qquad (70)$$

When the architectures operate in the word mode, Equation (70) reduces to

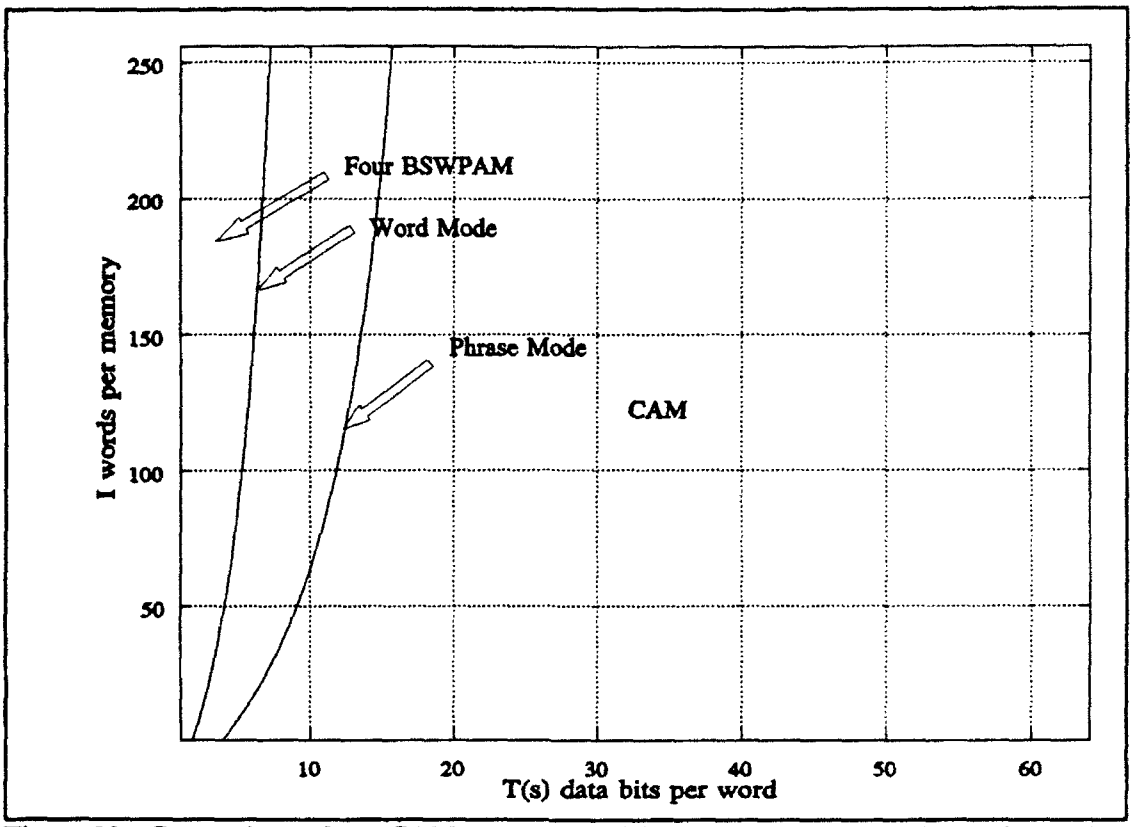

Figure 38. Comparison of the CAM to the Four BSWPAM Architecture in the Execution of BPI, RCI Operations.

$$\frac{0.906-1.38(T(s)+1)}{0.119} = I \tag{71}$$

When the architectures operate in the phrase mode, Equation (70) reduces to

$$\frac{0.906-1.38(T(s)+4)}{0.119} = I \tag{72}$$

Since Equations (70) and (71) yield negative values of I for all values of $T(s)$ in both modes, the CAM is always a better selection than the ESAM in the execution of BPI, RCI operation.

7.2.2.5 Comparison of the Single BSWPAM Architecture to the CAM Architecture in the Execution of BPD, RCI Operations. Let the time the single BSWPAM architecture requires to execute a BPD, RCI operation, denoted by Equation (38), equal the time required by the CAM architecture, denoted by Equation (36). Also let x equal 0, y equal

1, and z equal 0 to denote a BPD, RCI operation. The equality is fulfilled by the mathematical expression

$$(M(T(s)+T_{wdf}+T_{pf})+M-1)(7.96+0.11(T(s)+T_{wdf}+T_{pf})+0.06I) = (M(\frac{T(s)-1}{2})+M-1)(12.966+0.116(T(s)+T_{wdf}+T_{pf})+0.331I) \quad (73)$$

When the architectures operate in the word mode, Equation (73) reduces to

$$14.61+1.70T(s)+0.052T(s)^2 = 0.106(T(s)-2.137)I \quad (74)$$

The equation shows that I is negative when $T(s)$ is less than three and is positive otherwise. When the architectures operate in the phrase mode, Equation (73) reduces to

$$287.3+17.422T(s)+0.416T(s)^2 = 0.844(T(s)-1.595)I \quad (75)$$

The equation shows that I is negative when $T(s)$ is less than two and is positive otherwise.

Figure 39 shows the comparison of the single BSWPAM architecture to the CAM architecture in the execution of BPD, RCI operations. The figure, along with Equations (74) and (75), reveals that for memory lengths greater than approximately 50 words and for word lengths greater than 3 bits, the single BSWPAM is the better selection. For memory lengths less than about 50 words, the CAM may be the better selection depending upon the mode and the memory-array size. The CAM algorithm requires approximately one-half the number of steps as required by the single BSWPAM algorithm to execute the BPD, RCI operations, but the shorter algorithm-step execution time of the single BSWPAM architecture more than compensates for the algorithm disadvantage for memory lengths greater than approximately 50 words. The figure shows the execution-time penalty of having a word-select circuit perform the addressing as compared to the address-decode circuit.

7.2.2.6 Comparison of the Single BSWPAM Architecture to the ESAM Architecture in the Execution of BPD, RCI Operations. Let the time the single BSWPAM

architecture requires to execute a BPD, RCI operation, denoted by Equation (38), equal the time required by the ESAM architecture, denoted by Equation (45). The equality is fulfilled by the mathematical expression

$$(M(T(s)+T_{wdf}+T_{pf})+M-1)(7.96+0.11(T(s)+T_{wdf}+T_{pf})+0.06I) = (M(\frac{T(s)-1}{2})+M-1)(12.06+1.50(T(s)+T_{wdf}+T_{pf})+0.45I) \quad (76)$$

When the architectures operate in the word mode, Equation (76) reduces to

$$14.85+2.15T(s)-0.64T(s)^2 = 0.165(T(s)-1.73)I \quad (77)$$

When the architectures operate in the phrase mode, Equation (76) reduces to

$$273.42-5.88T(s)-5.12T(s)^2 = 1.32(T(s)-0.81)I \quad (78)$$

Equations (77) and (78) show that I is positive when $T(s)$ is greater than two but less than six. It also shows that for $T(s)$ greater than six bits, I is negative, which implies that the single BSWPAM architecture is the better selection.

Figure 40 shows the comparison of the single BSWPAM architecture to the ESAM architecture in the execution of BPD, RCI operations. The figure reveals that for both the word and phrase modes and for word lengths greater than about six bits, the single BSWPAM architecture is the better selection. In between one and six bits the better memory selection is determined by the mode and memory-array size.

The curves shown in Figures 39 and 40 are shaped the same except the curve in Figure 40 drops below the 0 words/memory threshold due to the negative coefficient in front of $T(s)^2$. The comparison of these two figures shows that increasing the time to execute a comparison using the CAM does not change the shape of the curve, but rather reduces the area under the curve which represents the memory-array sizes where the CAM is the better selection.

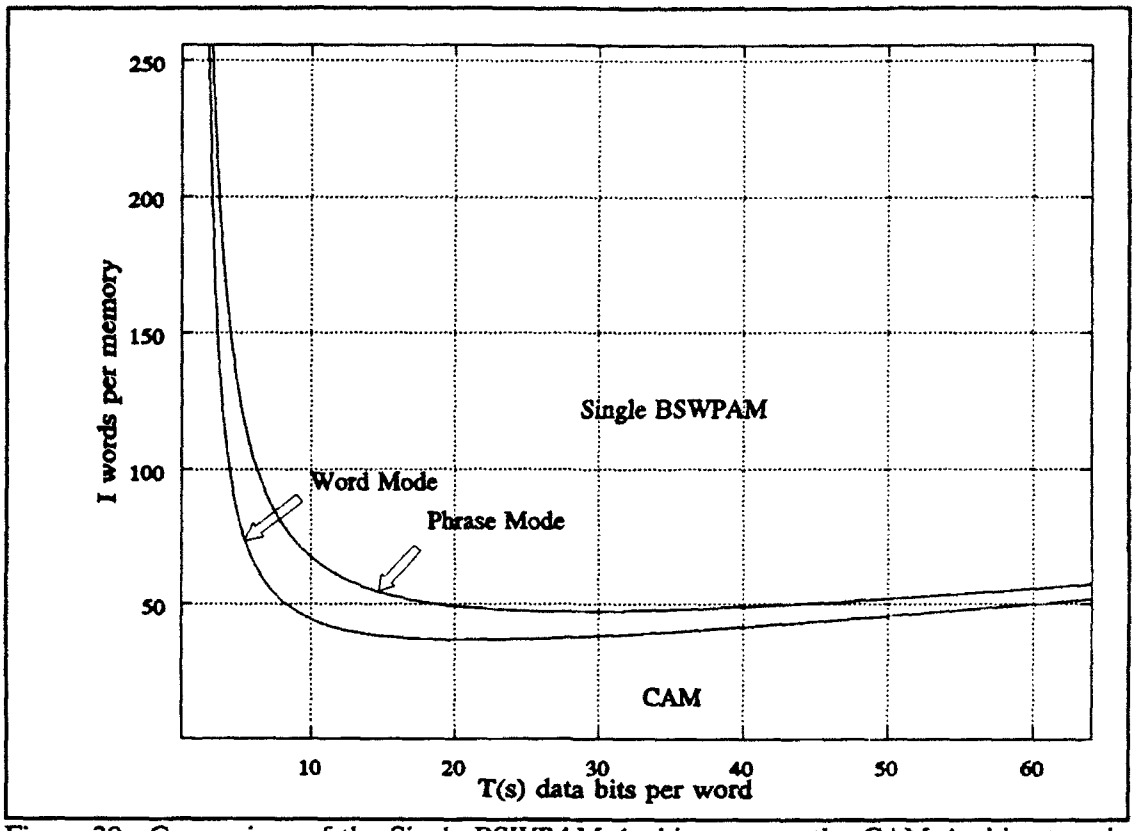

Figure 39. Comparison of the Single BSWPAM Architecture to the CAM Architecture in the Execution of BPD, RCI Operations.

7.2.2.7 Comparison of the ESAM Architecture to the CAM Architecture in the Execution of BPD, RCD Operations. Let the time the ESAM architecture requires to execute a BPD, RCD operation, denoted by Equation (60), equal the time required by the CAM architecture, denoted by Equation (36). Also let x equal 0, y equal 0, and z equal 1 to denote a BPD, RCD operation. The equality is fulfilled by the mathematical expression

$$\begin{aligned}(2M-1)(12.05+4.93(T(s)+T_{wdf}+T_{pf})+0.44I+0.09T(s)I) = \\ (M(T(s)+1)-1)(12.966+0.116(T(s)+T_{wdf}+T_{pf})+0.331I)\end{aligned} \tag{79}$$

When the architectures operate in the word mode, Equation (79) reduces to

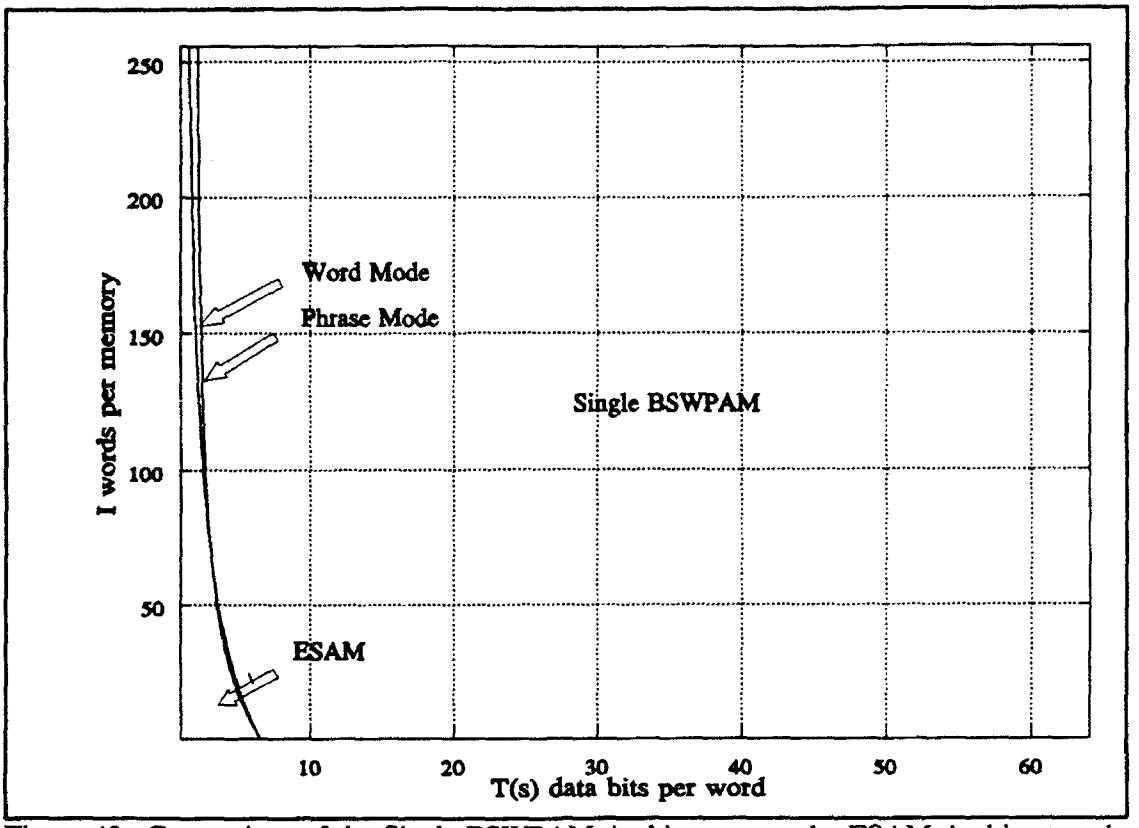

Figure 40. Comparison of the Single BSWPAM Architecture to the ESAM Architecture in the Execution of BPD, RCI Operations.

$$16.98-8.152T(s)-0.116T(s)^2 = 0.241I(T(s)-1.83) \tag{80}$$

The equation shows that I is negative for all values of $T(s)$ except 2. When $T(s)$ equals 2 and I is less than 2 then the CAM is the better selection; otherwise, the ESAM is the better selection. When the architectures operate in the phrase mode, Equation (79) reduces to

$$384.98-36.74T(s)-0.928T(s)^2 = 1.30(T(s)-3.30)I \tag{81}$$

The equation shows that I is negative for all values of $T(s)$ greater than 10.

Figure 41 shows the comparison of the ESAM to the CAM architecture in the execution of BPD, RCD operations. The figure reveals that for the word mode, and for the majority of memory-array sizes in the phrase mode, the ESAM is the better selection.

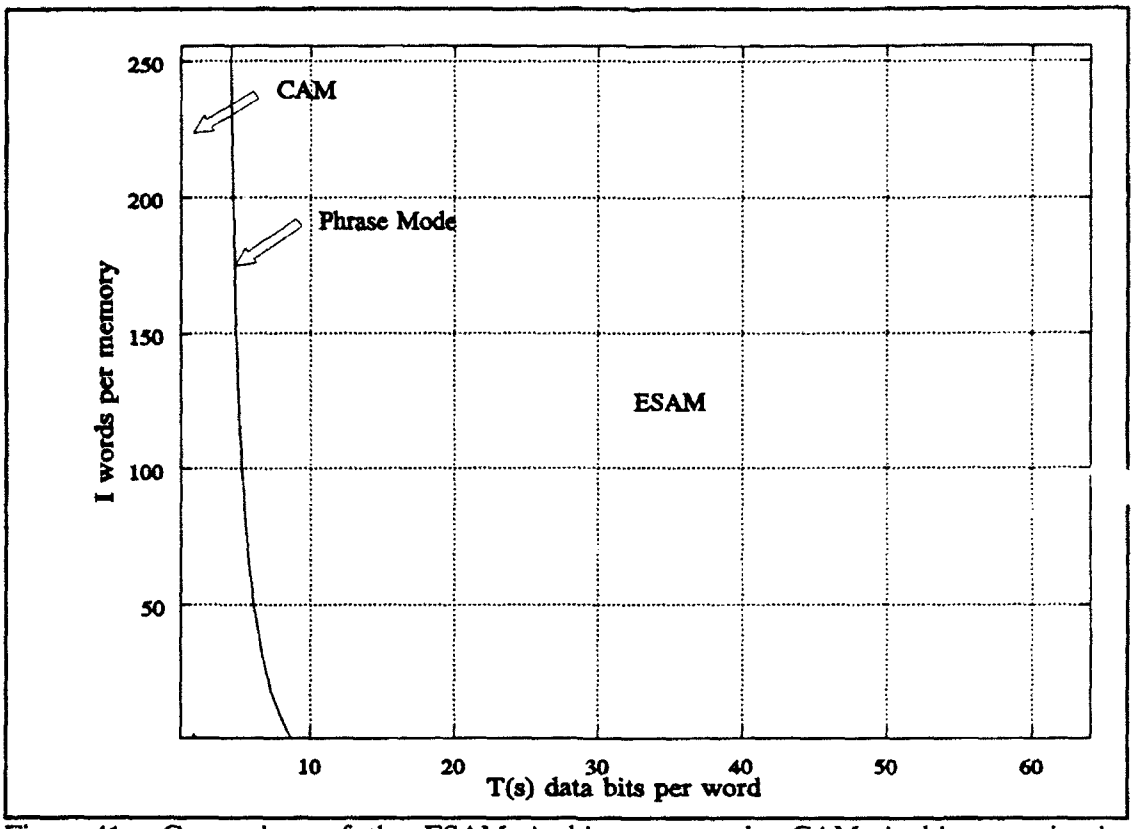

Figure 41. Comparison of the ESAM Architecture to the CAM Architecture in the Execution of BPD, RCD Operations.

7.2.2.8 Comparison of the ESAM to the Single BSWPAM Architecture in the Execution of BPD, RCD Operation. Let the time the ESAM architecture requires to execute a BPD, RCD operation, denoted by Equation (45), equal the time required by the single BSWPAM architecture, denoted by Equation (38). The equality is fulfilled by the mathematical expression

$$\begin{aligned}(2M-1)(12.05+4.93(T(s)+T_{wdf}+T_{pf})+0.44I+0.09T(s)I) = \\ (M(T(s)+T_{wdf}+T_{pf})+M-1)(7.96+0.11(T(s)+T_{wdf}+T_{pf})+0.06I)\end{aligned} \quad (82)$$

When the architectures operate in the word mode, Equation (82) reduces to

$$8.91-3.22T(s)-0.11T(s)^2 = -0.03(T(s)+12.66)I \quad (83)$$

The equation shows that I is positive for values of $T(s)$ greater than three. When the architectures operate in the phrase mode, Equation (82) reduces to

$$197.54-44.06T(s)-0.88T(s)^2 = -0.24(T(s)+33.04)I \quad (84)$$

Equation (84) is calculated by summing 8 extreme search execution times, 8 phrase and valid-data field equivalence search execution times, and 7 transfer execution times. The equation shows that I is positive for $T(s)$ greater than 5.

Figure 42 shows the comparison of the ESAM architecture to the single BSWPAM architecture in the execution of BPD, RCD operations. The figure reveals that for both the word and phrase modes, for data-field lengths greater than 5 bits, the ESAM architecture may be the better selection depending upon the mode and memory-array size.

7.2.2.9 Comparison of the CAM to the Single BSWPAM Architecture in the Word-Mode Execution of a 50% BPI, RCI and 50% BPD, RCI Instruction Mix. As an example of the strength of this analysis technique, suppose a program executes an associative-search operation mix that is 50% BPI, RCI and 50% BPD, RCI. Since the CAM architecture excels in executing BPI, RCI operations and the single BSWPAM architecture excels in executing BPD, RCI operations, these two architectures are selected for comparison. The analysis begins by equating the CAM architecture execution time to the single BSWPAM architecture execution time. This equation is defined by

$$\begin{aligned}&(M(\frac{T(s)+7}{4})-1)(12.966+0.116(T(s)+T_{wdf}+T_{pf})+0.331I) = \\ &(M(T(s)+T_{wdf}+T_{pf})+M-1)(7.96+0.11(T(s)+T_{wdf}+T_{pf})+0.06I)\end{aligned} \quad (85)$$

Suppose the analysis is performed in the word mode, which implies T_{pf} equals 0. Also suppose T_{wdf} equals 1. Equation (85) reduces to

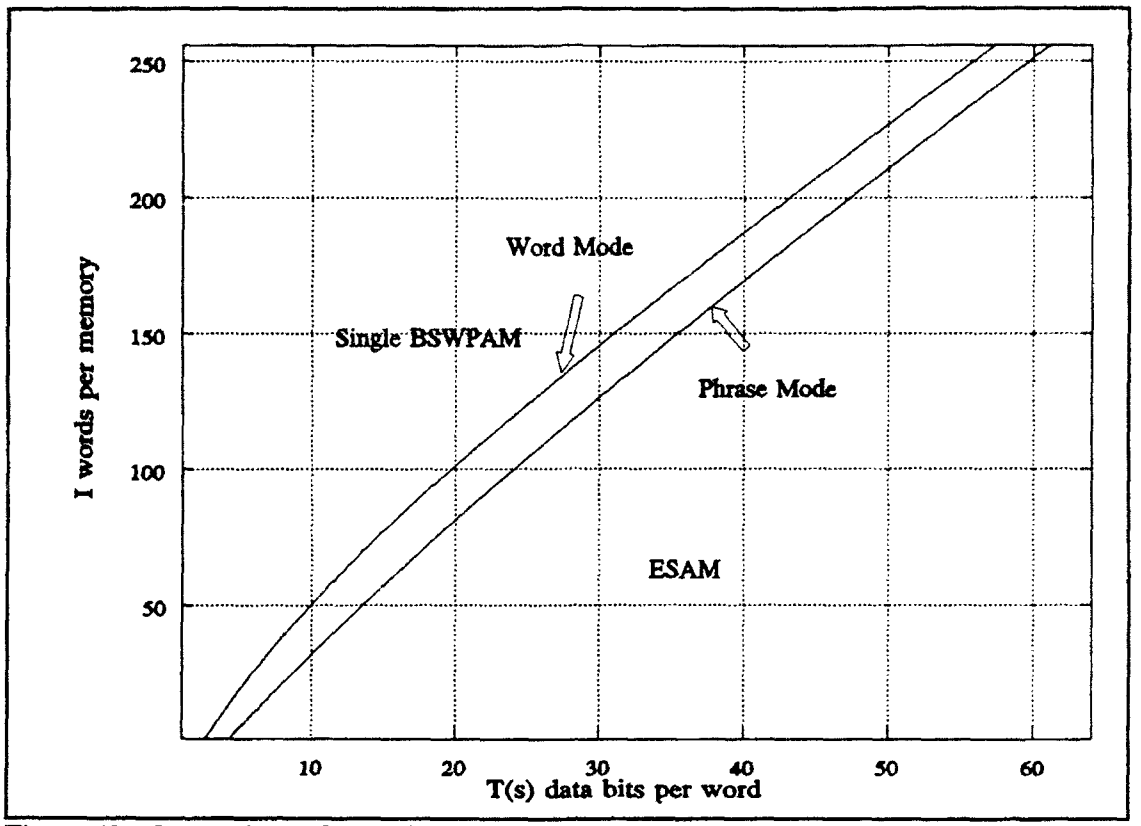

Figure 42. Comparison of the ESAM Architecture to the Single BSWPAM Architecture in the Execution of BPD, RCD Operations.

$$-1.738+4.826T(s)+0.082T(s)^2=0.023I(T(s)+8.174) \tag{86}$$

Figure 43 shows the comparison of the CAM to the single BSWPAM architecture. As the number of bits in a word increases, the CAM becomes a better selection for a larger number of memory-array sizes.

7.2.2.10 Discussion. The previous nine sections show a relationship between the hardware-influenced associative-search categories and the associative-memory organizations. Sections 7.2.2.1 through 7.2.2.4 showed that the CAM is the better selection for most memory-array sizes in executing BPI, RCI operations. Though the time to execute an algorithm step using a CAM is measured to be greater than the time to execute an algorithm step using each BSWPAM architecture, the superior algorithm of the CAM more than

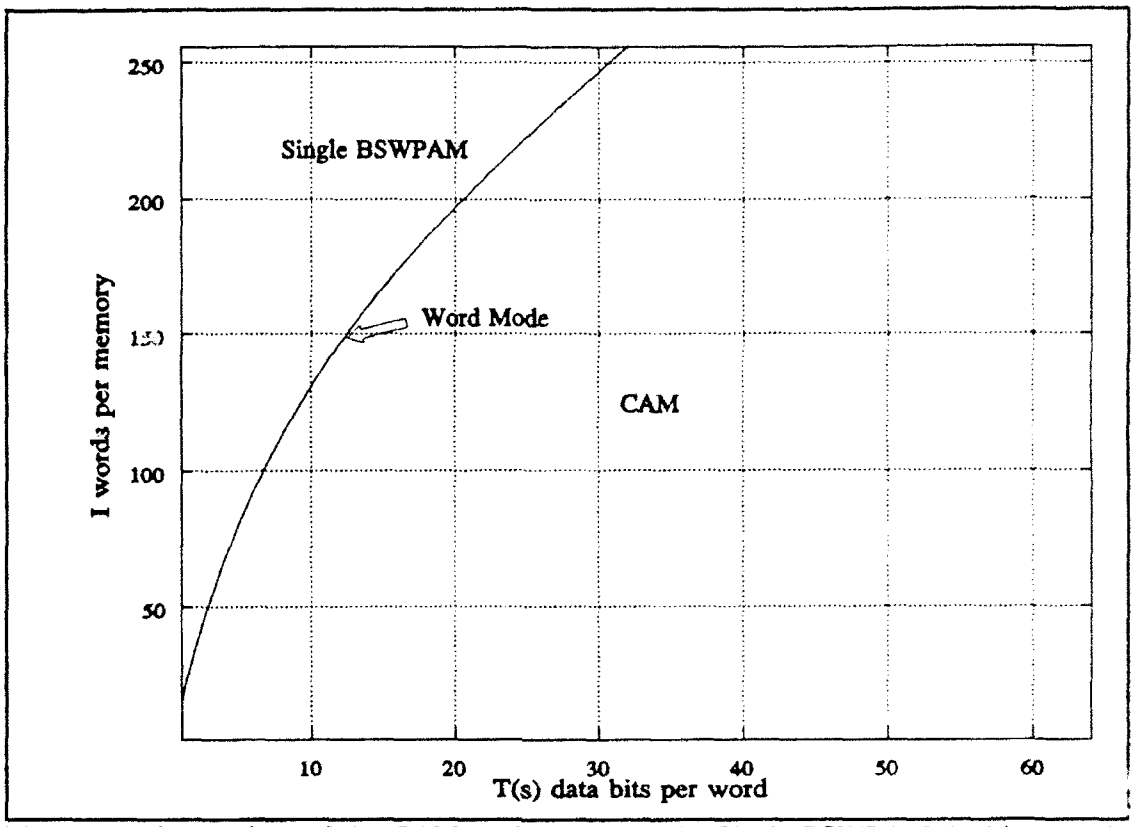

Figure 43. Comparison of the CAM Architecture to the Single BSWPAM Architecture in the Word Mode Execution of a *50% BPI, 50% BPD, RCI Operation Mix.*

compensates for its deficiency. From this analysis, it becomes evident why CAMs are the best associative-memory organization selection for applications such as look-up tables.

Sections 7.2.2.5 through 7.2.2.6 showed that the single BSWPAM architecture is the better selection for most memory-array sizes in executing BPD, RCI operations. The CAM requires about one-half the number of algorithm steps as compared to the single BSWPAM architecture in the execution of BPD, RCI operations, while the single BSWPAM architecture executes the step in a shorter time than the CAM architecture. The algorithm advantage of the CAM architecture dominates for memory lengths less than about 50 words, while the shorter step execution time of the single BSWPAM architecture makes it the better selection for larger memory lengths.

The CAM and the two BSWPAM architectures require about the same number of algorithm steps to execute a BPD, RCI operation, but the two BSWPAM architecture executes an algorithm step in a shorter time than the CAM architecture. Therefore, the two BSWPAM architecture is the better selection.

The four BSWPAM architecture requires about one-half the number of algorithm steps to execute a BPD, RCI operations as compared to the CAM architecture. It also executes the algorithm steps in a shorter time than the CAM architecture. So, the four BSWPAM architecture is the better selection.

Sections 7.2.2.7 and 7.2.2.8 showed that the ESAM executes BPD, RCD operations in a shorter time than the CAM. Again, the algorithm provides the advantage and not the algorithm-step execution time. In fact, the ESAM algorithm-step execution time is as much as an order of magnitude longer than any other architecture. The ESAM does not perform as well against the single BSWPAM. Though the ESAM algorithm requires fewer steps than the single BSWPAM, the superior algorithm-step execution time of the single BSWPAM architecture makes it more advantageous for a large number of memory array sizes.

This section has developed a decision criteria to select an associative-memory organization that minimizes the execution time of a mix of associative-search operations. By selecting the associative-memory architectures to compare and defining the instruction mix, a designer can plot the curves that show when one memory is better than another. Then by defining the memory-array size required for a problem, the designer can select the better organization. This analysis completes the first part of the dissertation purpose.

7.3 Memory-Layout Dimension Analysis

The previous section provided a decision criteria that determined the associative-memory architecture that executes a specific mix of associative-search instructions in the

shortest time, but this analysis alone is incomplete if the layout dimensions are an issue. This section addresses the second objective of the dissertation purpose statement by giving a memory-layout dimension analysis of each associative-memory architecture.

Associative-memory architectures require more area and are more complex than random-access memory architectures [12, 21, 22]. The layout dimensions depend upon the number of components and their circuit-layout dimensions, the amount of routing, and the designer's skill. This analysis attempts to minimize these variables. First, each memory uses a six-transistor SRAM for storage. Second, each memory uses the same metal layout scheme, where first layer metal is horizontal, second layer metal is vertical, and metal widths are 4 λ whenever possible. Third, the same person laid out each memory to reduce variations due to the designer's skill.

The section examines the SRAM, CAM, the single, two, and four BSWPAM, and ESAM architectures. The SRAM is examined for 8, 16, and 32-bit word lengths, while the associative-memory architectures are examined for 12, 20, and 36-bit word lengths. The associative-memory architectures require a valid-data field and a phrase field to properly operate, so four additional bits are required per word. These two additional fields account for the difference between the number of bits in a SRAM word and the number of bits in an associative-memory word.

7.3.1 SRAM-Layout Dimensions. This section calculates the SRAM-layout dimensions per bit of the architecture. Table 3 gives the layout dimensions for each component in the SRAM. The SRAM uses $\lceil \log_2 I \rceil$ drivers to form the address. It uses I decode circuits, J write and read circuits, and I times J SRAM cells. The SRAM was designed to eliminate metal runs between sub-components, so wiring does not add to the layout dimension. Equation (87) gives the SRAM-layout dimension per bit.

$$\frac{SRAM\ Size}{Bit} = (32\times50) + \frac{(32\times250)}{I} + \frac{(211\times50)}{J} + \frac{\lceil \log_2 I \rceil (32\times150)}{IJ} \qquad (87)$$

As can be seen from the equation, the most important component in the layout dimensions of the SRAM is the SRAM cell. For this reason, it received the most attention during the layout phase. A smaller SRAM cell design was laid out with dimension of 48 λ by 25 λ. This design was not selected for simulation for two reasons. First, the smaller design increased the load on the *data* and $\overline{data}$ ports which could in turn increase the time to read into or write from the memory array. Second, the smaller design required running ground in second level metal rather than first level metal. This violated the dissertation philosophy of using a consistent layout approach whenever possible.

7.3.2 CAM-Layout Dimensions. This section calculates the CAM-layout dimensions per bit of memory. Table 15 gives the component-layout dimensions. The CAM uses six control-1 drivers, two control-2 drivers, I match precharge circuits, $\lceil I/8 \rceil$ phrase-select circuits, J write and read circuits, and I times J CAM cells. The CAM was designed to eliminate metal runs between sub-components, so wiring does not add to the layout dimension. Equation (88) gives the CAM-layout dimensions per bit.

$$\frac{CAM\ Size}{Bit} = (36\times83) + \frac{(36\times437)}{I} + \frac{(33\times83)}{J} + \frac{\lceil \frac{I}{8} \rceil (803\times664) + 6(18\times104) + 2(32\times150)}{IJ} \qquad (88)$$

Once again, the most important component in the layout dimensions is the memory cell. Several CAM cell designs were considered. Grosspietsch shows a ten-transistor CAM cell [23]. It was rejected because the chosen design requires one less transistor and therefore should require a smaller layout dimension. Jones showed an eight-transistor pseudo-static CMOS CAM cell [9]. Though it may be smaller in design, it was rejected because of the necessity to be refreshed.

7.3.3 BSWPAM-Layout Dimensions. This section calculates the layout dimensions per bit of each BSWPAM. Table 31 gives the component-layout dimensions for each memory architecture. Each BSWPAM uses $\lceil log_2 I \rceil$ drivers to form the address. They use I decode circuits, slice-read circuits, processing elements, and address encoders, J write and read circuits, I times J memory cells, and the wiring that connects the bit-serial array to the PE array.

The next three equations calculate the layout dimensions for the single, two, and four BSWPAM architectures. Equation (89) gives the single BSWPAM architecture layout dimensions per bit.

$$\frac{SBSWPAM\ Size}{Bit} = (40\times60) + \frac{(40\times422)}{I} + \frac{(60\times274)+(112\times718)+(448\times I)}{J} + \frac{7(18\times104)+2(32\times150)+\lceil\log_2 I\rceil(150\times32)+\lceil\log_2 J\rceil(32\times150)}{IJ} \tag{89}$$

Equation (90) gives the two BSWPAM architecture layout dimensions per bit.

$$\frac{2BSWPAM\ Size}{Bit} = (40\times74) + \frac{(40\times417)}{I} + \frac{(74\times324)+(119\times945)+(952\times I)}{J} + \frac{7(18\times104)+2(32\times150)+\lceil\log_2 I\rceil(150\times32)+\lceil\log_2 J\rceil(32\times150)}{IJ} \tag{90}$$

Equation (91) gives the four BSWPAM architecture layout dimensions per bit.

$$\frac{4BSWPAM\ Size}{Bit} = (40\times99) + \frac{(40\times430)}{I} + \frac{(99\times407)+(276\times712)+(4416\times I)}{J} + \frac{7(18\times104)+2(32\times150)+\lceil\log_2 I\rceil(150\times32)+\lceil\log_2 J\rceil(32\times150)}{IJ} \tag{91}$$

The bit-serial cell has a large impact on each BSWPAM architecture layout dimension. Each memory used the same bit-serial cell, so the memory portion of the bit-serial cell dimensions were constant between the three architectures. The difference in the cell dimensions are

attributed to the *data-slice(i)* and $\overline{data}$*-slice(i)* ports that attach to the cell. The single BSWPAM requires a single *data-slice(i)* and $\overline{data}$*-slice(i)* port per word. The two BSWPAM requires two sets of *data-slice(i)* and $\overline{data}$*-slice(i)* ports per word. This added 14 λ to the cell height. The four BSWPAM requires four sets of *data-slice(i)* and $\overline{data}$*-slice(i)* ports per word. This added 39 λ to the cell height.

As can be seen from the equations, the wiring used to connect components becomes a major portion of the layout as *I* and *J* become large. In fact, the four BSWPAM architecture, which requires the most wire routing, is twice as large as any other associative-memory design. An eight BSWPAM architecture was designed, but its extensive routing made the circuit-layout dimensions so large that it was impractical.

7.3.4 ESAM-Layout Dimensions. This section calculates the ESAM-layout dimensions per bit of memory. Table 54 gives the component-layout dimensions. The ESAM uses six control-1 drivers, two control-2 drivers, *I* phrase-select circuits, *J* search-select, write, and read circuits, *I* times *J* extreme cells, and metal runs that connect the memory array to the word-select circuit. Equation (92) gives the ESAM-layout dimensions per bit.

$$\frac{ESAM\ Size}{Bit} = (64\times113) + \frac{(64\times637)}{I} + \frac{\lceil \frac{I}{8} \rceil (799\times664)}{IJ} + \frac{8(18\times104)+2(32\times150)}{IJ} + 1.35\frac{KI}{J} \quad (92)$$

The ESAM architecture and the CAM architecture are similar. The difference in size between the two is attributed to the ESAM-cell dimensions, which also affect the dimensions of the read, and write circuitry. The ESAM cell also forces routing between the ESAM array and the word-select circuitry. This routing becomes a large contributor to the memory layout dimensions as the number of words in the memory becomes large.

7.4 Execution-Time Versus Layout-Dimensions Analysis.

This section combines the associative-memory execution-time analysis of Section 7.2 and the layout dimension analysis of Section 7.3 to provide a more complete picture of the trade-offs a designer should consider when selecting an associative-memory architecture. The analysis evaluates each architecture in the execution of read, write, and associate-search instructions and operations.

Figures 44 and 45 show the read and write instruction execution time versus the memory-layout dimensions per bit for each architecture considered in this research. The numbers located beside each curve represent the number of words in the memory array for that simulation, and they increase in powers of two from 8 to 128 words. The SRAM is simulated for a 32-bit word length, while the associative-memory architectures are simulated for 36-bit word lengths.

The two figures reveal five points. First, and as expected, the SRAM executes read and write instructions in a shorter time and has smaller layout dimensions than any other architecture considered.

Second, the three BSWPAM architectures require about the same amount of time to execute read and write instructions, but the architectures are different in layout dimensions. Each BSWPAM uses the same circuitry for read and write instructions, so the execution times should be similar. The difference in layout dimensions is directly attributed to the circuitry necessary to execute associative-search instructions. As the number of bit-slices evaluated per algorithm step increases, so does the complexity and layout dimensions of the BSWPAM architectures.

Third, the CAM and ESAM architectures require about the same amount of time to execute read and write instructions, but the architectures are different in layout dimensions. Each architecture uses the same circuitry for read and write instructions, so the execution

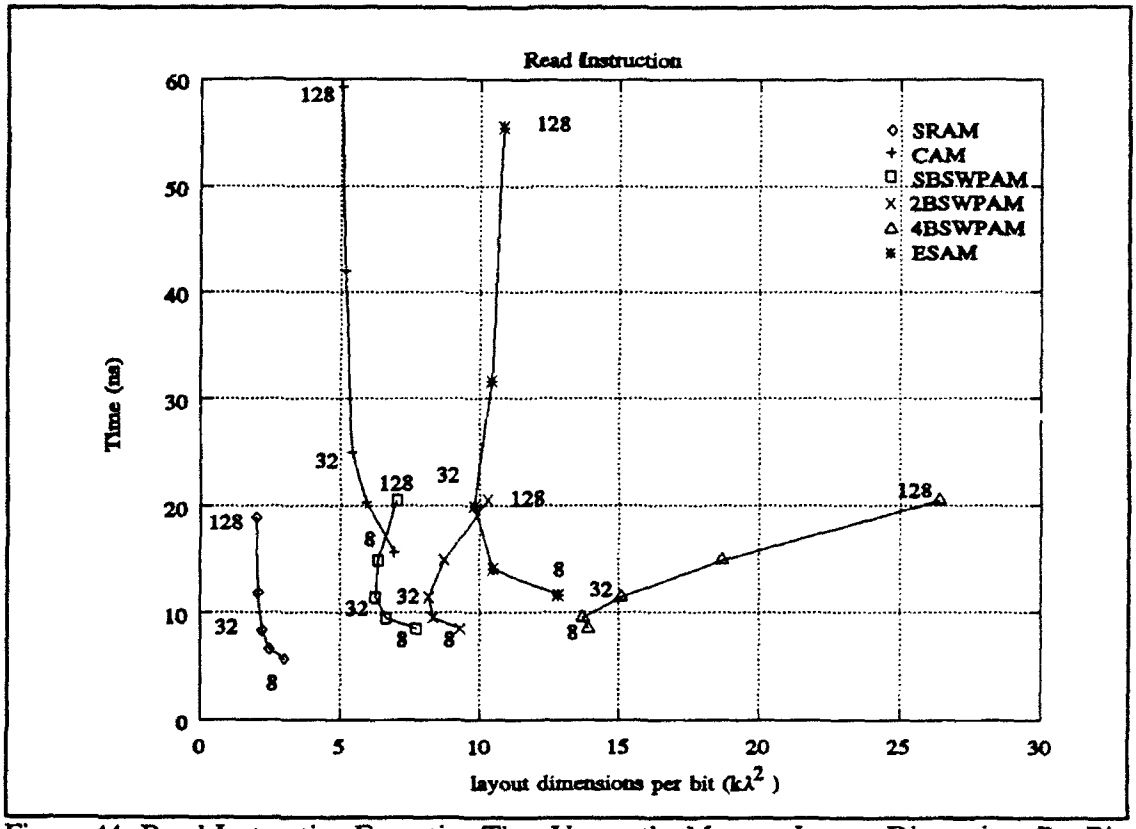

Figure 44. Read-Instruction Execution Time Versus the Memory Layout Dimensions Per Bit. The numbers within the figure represent the number of words within the memory array.

time should be similar. The difference in layout dimensions is predominantly caused by the differences in the memory array. The CAM cell requires about one-half the layout area of the more complex extreme cell.

Fourth, the CAM and ESAM read and write-instruction execution time increases with an increasing number of words within the memory array at a faster rate than the SRAM and each BSWPAM. This observation is attributed to the technique used to select words during the instruction. The CAM and ESAM uses a complex and large word-select circuit to select a memory word, while the SRAM and each BSWPAM uses a less complex and faster address-decode circuit.

Fifth, the layout dimensions of the associative-memory architectures that require routing between components may cause the layout dimensions per bit to increase, rather than

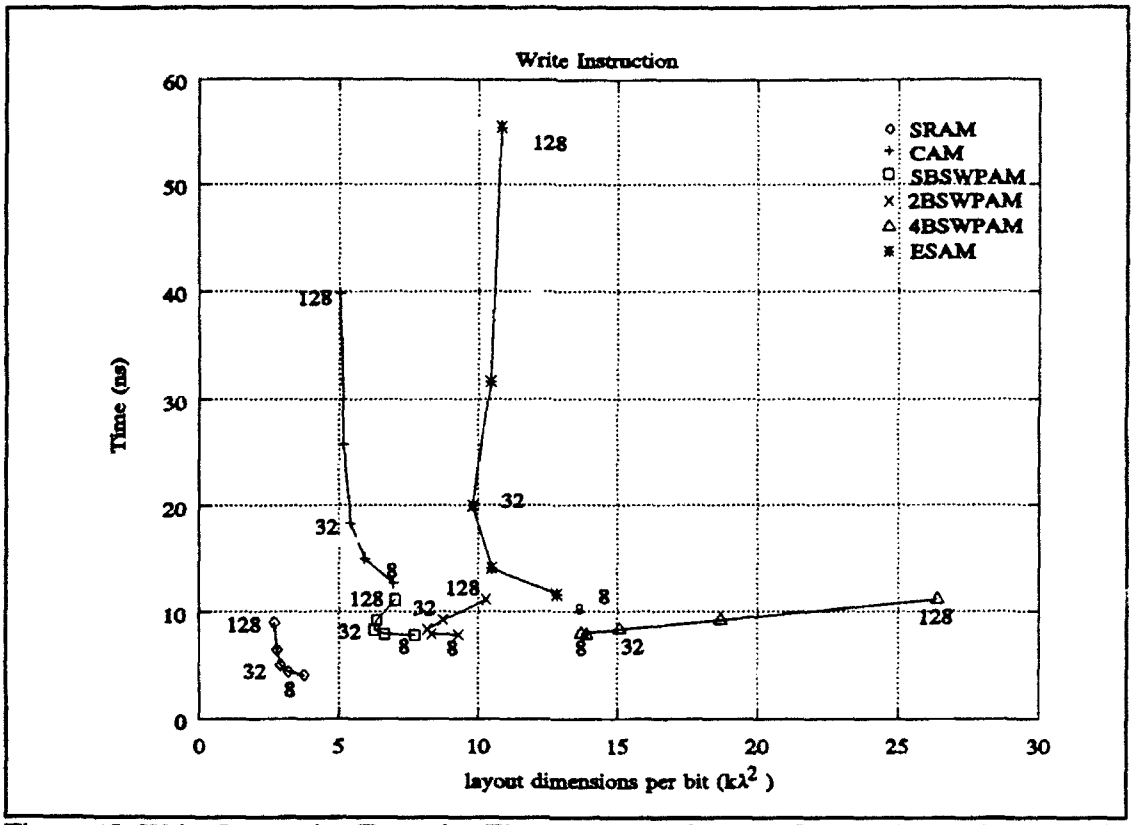

Figure 45. Write-Instruction Execution Time Versus the Memory Layout Dimensions Per Bit. The numbers within the figure represent the number of words within the memory array.

decrease, as the number of words in the memory array increases. Though the layout dimension overhead per word of components shared by multiple words goes down as the number of words in the memory array increases, the routing overhead increases. For architectures requiring extensive routing between components, like the four BSWPAM, the routing overhead dominates even at small memory-array lengths.

Figure 46 shows the BPI, RCI operation execution time versus the memory-layout dimensions per bit comparison for each associative-memory architectures considered in the research. The figure reveals three points. First, the CAM executes the associative-search operation in a shorter time and has a smaller layout dimension than any other associative-memory considered. Second, the ESAM has a comparable execution time, but has larger layout dimensions than either the CAM, the single or two BSWPAM architecture. The

algorithm gives the CAM and ESAM their execution-speed advantage. Third, the more complex the BSWPAM architecture, the shorter the execution time and the larger the layout dimensions.

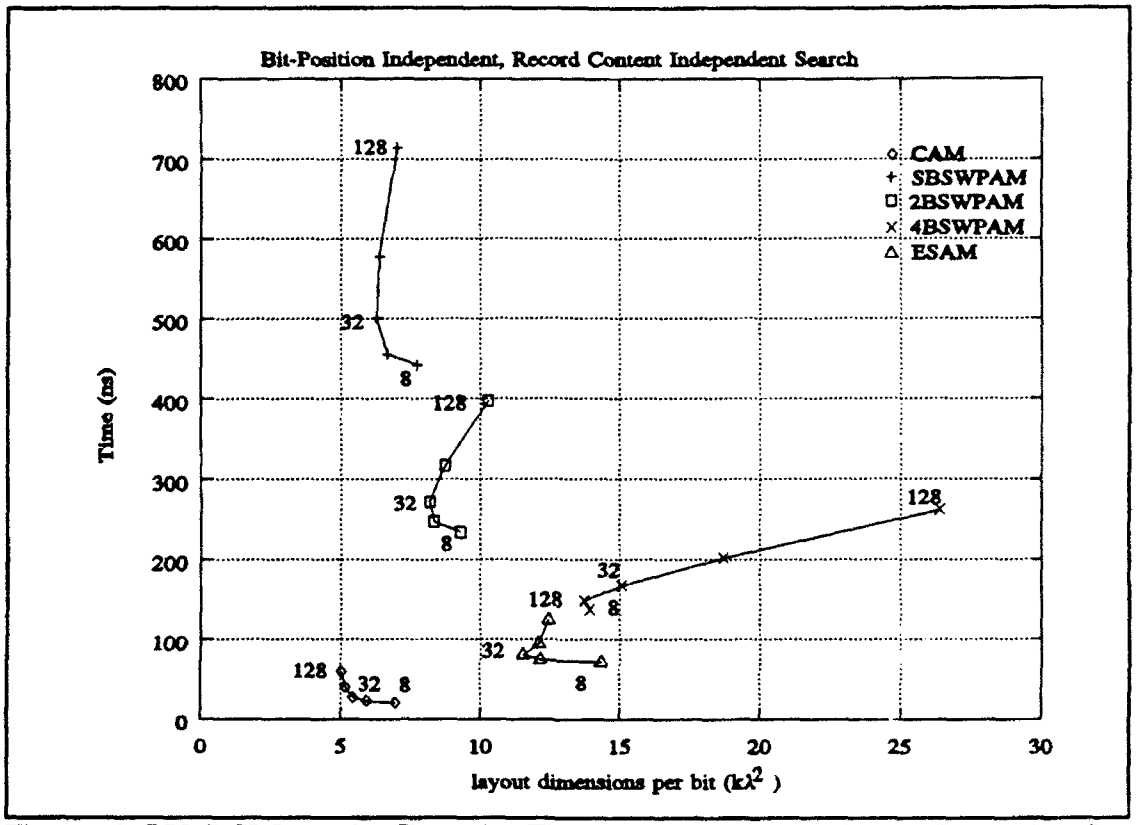

Figure 46. BPI, RCI Operation Execution Time Versus the Memory Layout Dimensions Per Bit.

Figure 47 shows the BPD, RCI operation execution time versus the memory-layout dimensions per bit for each associative-memory architecture considered in the research. The figure shows that the CAM, single and two BSWPAM architectures execute the BPD, RCI instructions in about the same amount of time and with approximately the same layout dimensions. The four BSWPAM executes the associative-search operation in a shorter time than any other associative-memory architecture, but is also larger than any other architecture. The ESAM is the worst of the associative-memory architectures in execution speed and is the second largest in layout dimensions per bit.

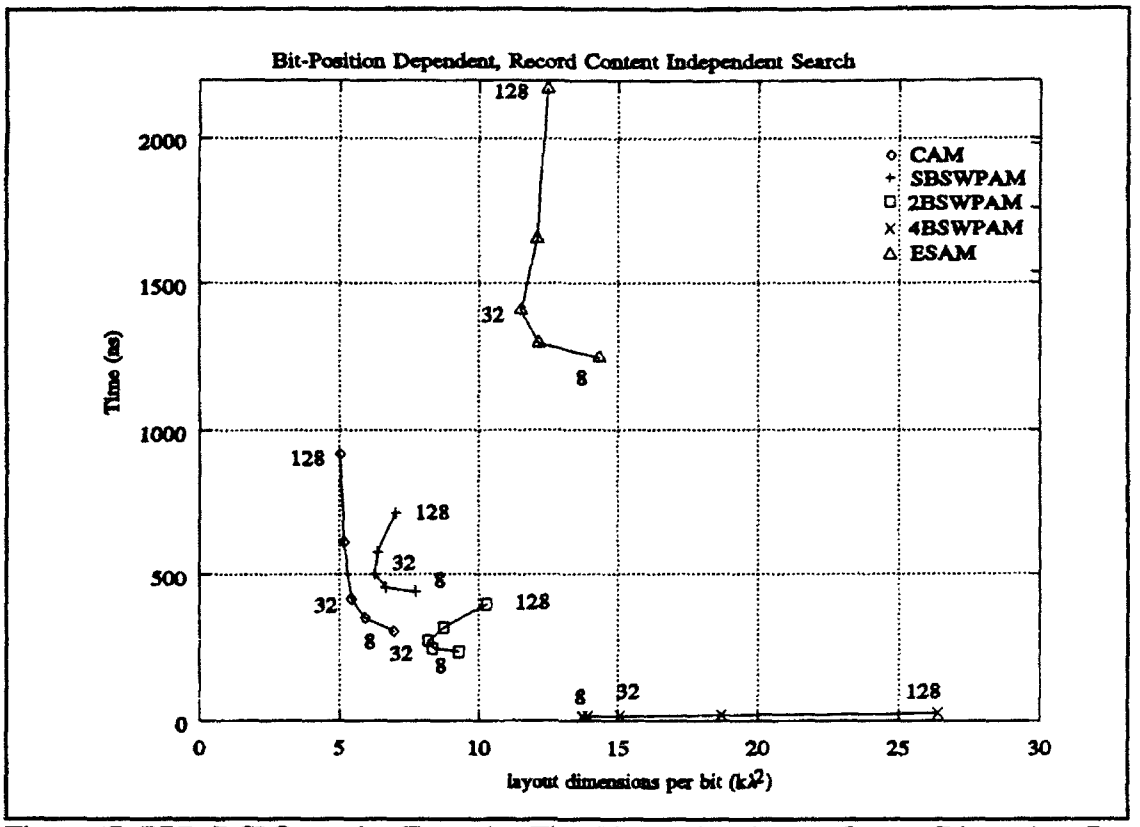

Figure 47. BPD, RCI Operation Execution Time Versus the Memory-Layout Dimensions Per Bit.

Figure 48 shows the BPD, RCD operation execution time versus the memory-layout dimension per bit comparison for each associative-memory architecture considered in the research. As expected, the ESAM executes the associative-search operations in a shorter time than the CAM architecture, and for most memory-array lengths in a shorter time than the BSWPAM architectures, though it was not the most compact. The single, two, and four BSWPAM architectures require approximately the same time to execute the associative-search operations due to similar algorithms. The CAM algorithm is similar to the BSWPAM, but the algorithm-step execution time disadvantage previously discussed makes it the worst performer.

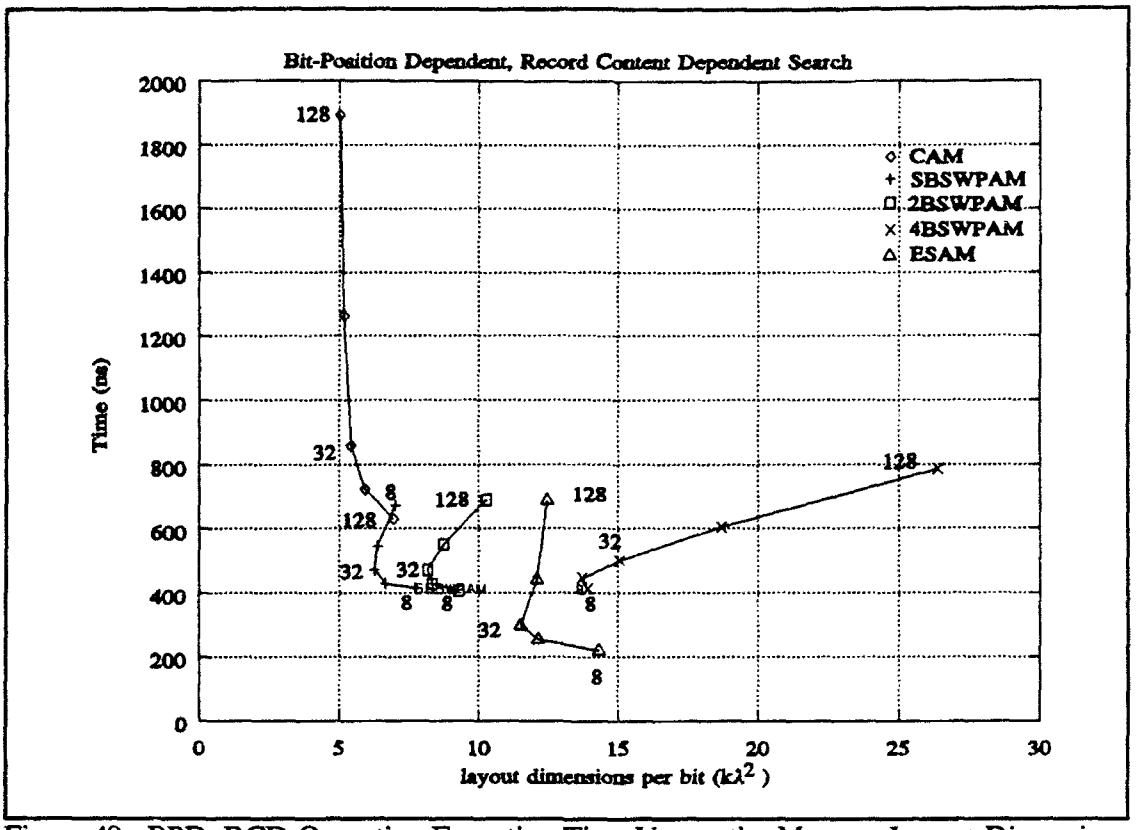

Figure 48. BPD, RCD Operation Execution Time Versus the Memory-Layout Dimensions per Bit.

7.5 Summary and Conclusion

Chapter VII completes the dissertation purpose as stated in Section 1.3. Section 7.2.1 used the measured data of Chapters III through VI to determine the read and write-instruction execution times of each architecture. The results show that addressed memory architectures execute read and write instructions in a shorter time than architectures using word-select circuitry to internally select an address. The results also show that adding bits to a word less impacts the read and write-instruction execution time less than adding words to the memory-array length. This implies that the optimal memory-array size for an associative-memory architecture in executing read and write instructions will place at least one record per word.

Section 7.2.2 gave an instruction execution-time analysis. It used the quantitative analysis of Chapter II and the measured data of Chapters IV through VI to show a relationship between the associative-search operations and the associative-memory organizations. The analysis confirms that the CAM executed BPI, RCI operations in a shorter time than each BSWPAM architecture for most memory-array sizes. The analysis also confirms that the single BSWPAM architecture executes BPD, RCI operations in a shorter time than the CAM and ESAM architectures for most memory-array sizes. Finally, the analysis confirms that the ESAM executes BPD, RCD operations in a shorter time than the CAM or the BSWPAM architectures for most memory-array sizes.

Section 7.3 provided a memory-layout dimension analysis using the component size information from Chapters III through VI. This analysis provides the memory designer a rule-of-thumb criteria for memory-array size decisions. Section 7.4 combined the information provided in Sections 7.2 and 7.3 to provide an execution-time versus layout-dimensions analysis. From the analysis, it was determined that the CAM is the most compact associative-memory architecture design, but the single BSWPAM architecture is comparable. The two BSWPAM architecture is approximatcly twice as large as the CAM, but its superior algorithm execution of BPD, RCI operations makes it attractive. The four BSWPAM offers superior instruction execution times of most associative-search operation mixes, but the layout dimensions indicate that it is the most costly in size. The ESAM provides superior instruction execution times when compared to the CAM, but does not compare as well to the BSWPAM. Except for the four BSWPAM, the architecture layout dimensions are also larger.

VIII. Conclusion

This research endeavor developed a decision criteria to select an associative-memory organization that minimizes the execution time of a mix of associative-search operations. To accomplish this objective, the research endeavor mathematically expressed Feng's associative-search operations using operators executable by an associative-memory organization. From the mathematical expressions, a hardware-influenced reclassification of Feng's associative-search operations became evident. The three categories are: bit-position independent (BPI), record-content independent (RCI); bit-position dependent (BPD), RCI; and BPD, record-content dependent (RCD) associative-search operations. Next, the research endeavor performed a quantitative analysis of a variety of associative-memory organizations to determine the algorithms used to execute the hardware-influenced associative-search operations. These algorithms are functions of the number of bits in a field, the instruction mix, and the algorithm-step execution time.

The research continued by designing an architecture representation of the content-addressable memory (CAM), and extreme-search associative-memory (ESAM) organization, and three architecture representations of the bit-serial word-parallel associative-memory (BSWPAM) organization, entitled single, two, and four BSWPAM. These designs were functionally verified using VHDL, laid-out using Magic, and simulated, using SPICE, to obtain the algorithm-step execution times. The algorithm-step execution times were collected for a variety of memory-array sizes ranging from 8 to 36-bit words and from 4 to 128-word memory arrays. The data was used to develop characteristic equations for the algorithm-step execution times, which can be multiplied by the number of algorithm steps required to execute an operation to determine the instruction execution time.

Combining the algorithms and the algorithm-step execution-time equations gave a unified equation dependent upon the number of bits in a word, the number of words in a memory array, and the instruction mix. The unified equation for each architecture is compared to determine the best associative-memory organization to execute a specific mix of associative-search operations, thus completing the research purpose.

These unified equations were compared for a variety of operation mixes. The results indicate that the CAM architecture executes BPI, RCI operations in a shorter time than the single, two, and four BSWPAM architecture for most memory-array sizes because of an algorithm advantage. The CAM architecture executes BPI, RCI operations in a shorter time than the ESAM architecture because of an algorithm-step execution-time advantage.

The single BSWPAM architecture executes BPD, RCI operations in a shorter time than the CAM architecture for most memory-array sizes. Even though the CAM requires about one-half the number of algorithm steps, the single BSWPAM executes the algorithm step in a shorter time than the CAM. For most memory-array sizes the execution-time advantage more than compensates for the algorithm disadvantage. The two BSWPAM architecture executes BPD, RCI operations in a shorter time than the CAM architecture for most memory-array sizes. The CAM and the two BSWPAM architectures require about the same number of algorithm steps to execute a BPD, RCI operation, but the two BSWPAM architecture executes an algorithmic step in a shorter time than the CAM. The four BSWPAM architecture executes BPD, RCI operations in a shorter time than the CAM architecture for most memory-array sizes. The four BSWPAM architecture requires about one-half the number of algorithm steps to execute a BPD, RCD operation as compared to the CAM. It also executes the algorithmic steps in a shorter time than the CAM.

The ESAM architecture executes BPD, RCD operations in a shorter time than the CAM architecture for most memory-array sizes, and it executes BPD, RCD operations in a

shorter time than the BSWPAM architectures for a large number of memory array sizes. Again, the algorithm provides the advantage and not the algorithm-step execution time. In fact, the ESAM algorithm-step execution time is as much as an order of magnitude longer than any other architecture.

Because the research endeavor designed the layouts necessary to fabricate each architecture, the memory-layout dimensions were measurable. This data was combined with the unified equations to demonstrate tradeoffs between execution time and memory-layout dimensions.

The analysis revealed that the CAM architecture executes BPI, RCI operations in a shorter time and has a smaller layout dimension than the other architectures. The number of bit-slices a BSWPAM architecture compares is inversely proportional to the execution time of BPI, RCI and BPD, RCI associative-search operations and directly proportional to the layout dimensions. On the other hand, the number of bit-slices a BSWPAM architecture compares has no affect upon the read and write-instruction execution time. The ESAM architecture executes BPD, RCD instructions in a shorter time than the CAM architecture, but has a large memory array that gives it the second largest layout dimensions, where the four BSWPAM architecture has the largest layout dimension

In support of developing a decision critieria to select the best associative memory to execute a mix of instructions, the research endeavor completed five analyses not previously published. First, the mathematical expressions of Feng's associative-search operations have not been previously published. Second, the reclassification of Feng's associative-search operations into three hardware-influenced categories is defined from those mathematical expressions. Third, several associative-search algorithms necessary to execute the three hardware-influenced associative-search operations were defined while performing the quantitative analysis. Fourth, the combining of the algorithms and the algorithm-step

execution times to develop a criteria to select the best architecture to execute an instruction mix has not been published. Finally, the execution-speed versus layout-dimension analysis is new.

Appendix A: The Average Number of BPI, RCI Instructions a CAM Requires to Execute a BPD, RCI Operation

This appendix derives the average number of BPI, RCI instructions a CAM requires to execute a BPD, RCI operation. The derivation assumes an equal probability exists that a datum will be found within the comparand or database. Let $N \in \mathbf{N}$ be the maximum number of bits in a field, and let $n \in \mathbf{N}$ be a field length where $1 \le n \le N$. Let $M=2^N$, and $m=2^n$. Denote an element in an M by N matrix, A, with $a_{m,n}$. The element, $a_{m,n}$, is the number of BPI, RCI instructions a CAM requires to execute a BPD, RCI operation for a specific comparand value equal to $m-1$ and a comparand field length equal to n. The average number of BPI, RCI instructions a CAM requires to execute a BPD, RCI operation for an n-bit field length is derived by Equation (93).

$$AVG_n = \frac{\sum_{k=1}^{2^n} a_{k,n}}{2^n} \tag{93}$$

The appendix addresses BPD, RCI operations, but will examine only the less-than operation. By symmetry the less-than operation requires the same number of BPI, RCI instructions as the greater-than operation. Also, the less-than operation of one comparand value takes the same amount of BPI, RCI instructions as the less-than or equal-to operation of the comparand value minus one. Therefore, the less-than operation characterizes the BPD, RCI operations.

Suppose a database contains a single-bit field. Also, suppose the comparand equals 0. Since no field value is less than 0, $a_{1,1}$ equals 0. Now, suppose the comparand value equals 1. The CAM requires a single BPI. RCI instruction to find all field values less than 1, so $a_{2,1}$ equals 1. Therefore, AVG_1 equals ½.

A two-bit comparand field can be thought of as two repetitions of a single-bit field with the first half of the possible two-bit comparand values formed by appending the single-bit

field elements with a 0 prefix. The second half of the two-bit comparand values are formed by appending the single-bit field elements with a 1 prefix. In general, $(n+1)$-bit field is two repetitions of an n-bit field appended with either a 0 or a 1 prefix.

The number of BPI, RCI instructions required to execute a less-than operation on an $(n+1)$-bit field can be determined from the n-bit field using five equations. Equation (94) shows that the number of BPI, RCI instructions required to search the first ¼ of the possible comparand values for an $(n+1)$-bit field equals the number of BPI, RCI instructions for the first ½ of the possible comparand values for an n-bit field.

$$\sum_{k=1}^{2^{n-1}} a_{k,n+1} = \sum_{k=1}^{2^{n-1}} a_{k,n} \tag{94}$$

Equation (95) shows that it takes one BPI, RCI instruction to find all elements less than a comparand equal to 2^{n-1} on an $(n+1)$-bit field. This instruction finds the subset of all comparand values with a 0 in the most and second most-significant bit position.

$$a_{2^{n-1}+1,n+1} = a_{2^{n-1},n} = 1 \tag{95}$$

Equation (96) shows that adding a most-significant bit equal to 0 to an n-bit field length between comparand values of $2^{n-1}+1$ and 2^n-1 adds a single BPI, RCI instruction to each element in the $(n+1)$-bit field within the same range of comparand values.

$$\sum_{k=2^{n-1}+2}^{2^n} a_{k,n+1} = \sum_{k=2^{n-1}+2}^{2^n} (a_{k,n}+1) \tag{96}$$

Equation (97) shows that it takes one BPI, RCI instruction to find all elements less than a comparand equal to 2^n on an $(n+1)$-bit field. This instruction finds the subset of all comparand values with a 0 in the most-significant bit position.

$$a_{2^n+1,\,n+1} = a_{2^{n-1},\,n} = 1 \tag{97}$$

Equation (98) shows that adding a most-significant bit equal to 1 to an n-bit field length between comparand values of 2^n+1 and $2^{n+1}-2^{n-1}-1$ adds a single BPI, RCI instruction to the $(n+1)$-bit field within the same range of comparand values.

$$\sum_{k=2^n+2}^{2^{n+1}-2^{n-1}} a_{k,\,n+1} = \sum_{k=2^{n-1}+2}^{2^n} a_{k,\,n}+1 \tag{98}$$

Equation (99) shows that the number of BPI, RCI instructions required to search the last ¼ of the possible comparand values for an $(n+1)$-bit field equals the number of BPI, RCI instructions for the second ½ of the possible comparand values for an n-bit field.

$$\sum_{k=2^{n+1}-2^{n-1}+1}^{2^{n+1}} a_{k,\,n+1} = \sum_{k=2^{n-1}+1}^{2^n} a_{k,\,n} \tag{99}$$

The total number of BPI, RCI instructions required to execute a less-than operation for an $(n+1)$-bit field and in terms of an n-bit field is denoted by Equation (100).

$$\sum_{k=1}^{2^{n+1}} a_{k,\,n+1} = \sum_{k=1}^{2^n} a_{k,\,n}+\left(2\sum_{k=2^{n-1}+1}^{2^n} a_{k,\,n}\right)+2^n-2 \tag{100}$$

The column elements of A are nearly symmetric between the comparand values with a 0 most-significant bit and the comparand values with a 1 most-significant bit. The symmetric relationship is defined by Equation (101).

$$\sum_{k=1}^{2^n} a_{k,\,n}+1 = 2\sum_{k=2^{n-1}+1}^{2^n} a_{k,\,n} \tag{101}$$

Substituting Equation (101) into Equation (100) yields Equation (102).

$$\sum_{k=1}^{2^{n+1}} a_{k,\,n+1} = 2\sum_{k=1}^{2^n} a_{k,\,n}+2^n-1 \tag{102}$$

At this point, the analysis will show, with an inductive proof, that Equation (103) is true.

$$\sum_{k=1}^{2^{n-1}} a_{k,n} = (n-1)2^{n-1}+1 \tag{103}$$

This equation was derived from an analysis of 1, 2, 3, 4, 5, and 6-bit field lengths. The appendix has shown the equation to be true when $n=1$. Suppose the equation is true for value $n=j$, the proof is completed by showing that it is also true for $n=j+1$. Substituting Equation (103), where $n=j$, into Equation (102) yields Equation (104).

$$\sum_{k=1}^{2^{n+1}} a_{k,j+1} = 2((j-1)2^{j-1}+1) + 2^j - 1 = j2^j + 1 \tag{104}$$

Since Equation (104) shows Equation (103) is true for $n=j+1$, the inductive proof is complete. Using Equation (104), AVG_n is defined by Equation (105).

$$AVG_n = \frac{(n-1)2^{n-1}+1}{2^n} = \frac{n-1}{2} + \frac{1}{2^n} \tag{105}$$

Appendices B - G: VHDL Code

Due to its size, Appendices B - G were not attached to this report. A copy of these appendices are maintained in the AFIT VLSI Laboratory. For information on obtaining a copy of this code, send correspondence to or call Major Mark Mehalic at:

Air Force Institute of Technology
AFIT/ENG, 2950 P St.
Wright-Patterson AFB, OH 45433-7765
Phone (513) 255-3576
DSN 785-3576

References

1. J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach.* San Mateo, California: Morgan Kaufmann Publishers, Inc., 1990.

2. T. Ogura, S. Yamada, and J. Yamada, " A 20-Kbit CMOS Associative Memory LSI for Artificial Intelligence Machines," *Proceedings IEEE International Conference on Computer Design: VLSI in Computers*, pp. 574-577, 1986.

3. T. Yeap, W. Loucks, W. Snelgrove, and S. Zaky, "Implementing the VASTOR Architecture Using a VLSI Array of 1-Bit Processors," *IEEE International Conference on Computer Design*, pp. 494-499, 1985.

4. G. Anderson and R. Kain, "A Content-Addressed Memory Designed f r Database Applications," *International Conference on Parallel Processing*, pp. 191-195 August 24-27, 1976.

5. C. DiFiore and P. Berra, "A Quantitative Analysis of the Utilization of Associative Memories in Database Management," *IEEE Transactions on Computers, vol. C-23, no. 2*, pp. 121-132 , February 1974.

6. T. Ogura, S. Yamada, and T. Nikaido, "A 4-Kbit Associative Memory LSI," *IEEE Journal Solid-State Circuits, vol. SC-20*, pp. 1277-1282, Dec. 1985.

7. H. Kadota, J. Miyake, Y. Nishimichi, H. Kudoh, and K. Kagawa, "An 8-Kbit Content-Addressable and Reentrant Memory," *IEEE Journal of Solid-State Circuits, vol. SC-20*, pp. 951-957, Oct. 1985.

8. S. Jones, "Design, Selection and Implementation of a Content-Addressable Memory for a VLSI CMOS Chip Architecture," *IEE Proceedings, vol. 135, pt. E, no. 3*, pp. 165-172, May 1988.

9 S. Jones, I. Jalowiecki, S. Hedge, and R. Lea, " A 9-KBit Associative Memory for High-Speed Parallel Processing Applications," *IEEE Journal of Solid-State Circuits, vol. 23, no. 2*, pp. 543-548, April 1988.

10. B. Parhami, "Associative Memories and Processors: An Overview and Selected Bibliography," *Proceedings of the IEEE, vol. 61, no. 6*, pp. 722-730, June 1973.

11. J. Armstrong, *Chip-Level Modeling with VHDL.* Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1989.

12. K. Hwang and F. Briggs, *Computer Architecture and Parallel Processing.* New York: McGraw-Hill Company, 1984.

13. N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*," Massachusetts: Addison-Wesley, 1985.

14. A. Falkoff, "Algorithms for Parallel-Search Memories," *Journal of the ACM, vol 9*, pp. 488-511, October 1962.

15. W. Davis and D. Lee, "Fast Search Algorithms for Associative Memories," *IEEE Transactions on Computers, vol. C-35, no. 5*, pp. 456-461, May 1986.

16. L. Chisvin and R. Duckworth, "Content-Addressable and Associative Memory: Alternatives to the Ubiquitous RAM," *Computer*, pp. 51-63, July 1989.

17. Meta-Software, HSPICE User's Manual, H9001, Meta-Software, Inc., 1300 White Oaks Road, Campbell, CA 95008, 1991.

18. C. Mead, *Analog VLSI and Neural Systems*. Massachusetts: Addison-Wesley, 1989.

19. E. Frei and J. Goldberg, "A Method for Resolving Multiple Responses in a Parallel Search File," *IRE Transactions on Electronic Computers, vol EC-10*, pp. 718-722, Dec 1961.

20. C. Mansel, Design and Analysis for the Arithmetic Content-Addressable Memory. MS Thesis AFIT/GE/ENG/92D-24. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1992.

21. J. Hayes, *Computer Architectures and Organizations*. New York: McGraw-Hill Company, 1988.

22. A. DeCegama, *The Technology of Parallel Processing: Parallel Processing Architectures and VLSI Hardware Volume 1*. Englewood Cliffs, New Jersey: Prentice-Hall, 1989.

23 K. Grosspietsch, "Associative Processors and Memories: A Survey," *IEEE Micro*, pp. 12-19, June 1992.

VITA

Captain David Wayne Banton was born October 15, 1956, in Covington, Virginia. Upon graduating from high school in 1975, he enlisted into the USAF. In 1979, Sergeant Banton received an honorable discharge and began an undergraduate degree. In 1982, he received the degree of Bachelors of Science in Engineering from the University of Central Florida, in Orlando, Florida. Later that year, he received a commission in the USAF through the Air Force Officer Training School. In 1986, he received the degree of masters of Science in Electrical Engineering from the Air Force Institute of Technology, Wright Patterson AFB, Ohio. In 1990, he began his Ph D program at the Air Force Institute of Technology. In 1991, he received his candidacy.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704 0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 93	Dissertation

4. TITLE AND SUBTITLE
A Decision Criteria to Select an Associative-Memory Organization that Minimizes the Execution Time of a Mix of Associative-Search Operations.

5. FUNDING NUMBERS

6. AUTHOR(S)
David W. Banton

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB, OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/DS/ENG/93-02

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Captain Daniel Zahirniak
Wright Laboratory, AAWP
WPAFB, OH 45433-6534

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The dissertation develops a decision criteria to select an associative-memory organization that minimizes the execution time of a mix of associative-search operations and a decision criteria to estimate the layout dimensions of each organization for a specified memory size. The dissertation reclassifies Feng's associative-search operations into three hardware-influenced categories: bit-position independent (BPI), record-content independent (RCI); bit-position dependent (BPD), RCI; and BPD, record-content dependent (RCD). It develops a relationship between the categories and three associative-memory organizations. the CAM, the bit-serial word-parallel associative memory (BSWPAM), and the extreme-search associative memory (ESAM). A version of the CAM, three versions of the BSWPAM, and a version of the ESAM organizations were designed and simulated to show that for most memory sizes, BPI, RCI operations require less time when executed on a CAM, BPD, RCI operations require less time when executed on a BSWPAM organization, and for many memory sizes, BPD, RCD operations require less time when executed on an ESAM. The dissertation calculates the layout dimensions of each memory. The results indicate the CAM is the most area efficient followed by the single, and two BSWPAM, the ESAM, and the four BSWPAM.

14. SUBJECT TERMS	15. NUMBER OF PAGES
associative memory, content-addressable memory, bit-serial word-parallel associative memory	203
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	